SPEAR PHISHING ATTACK DETECTION

THESIS

David T. Merritt, Captain, USAF

AFIT/GCE/ENG/11-05

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCE/ENG/11-05

SPEAR PHISHING ATTACK DETECTION

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science in Computer Engineering

David T. Merritt, BSCE

Captain, USAF

March 2011

AFIT/GCE/ENG/11-05

SPEAR PHISHING ATTACK DETECTION

David T. Merritt, BSCE
Captain, USAF

Approved:

Barry Mullins	1 Mar 11
Dr. Barry E. Mullins (Chairman)	Date
Richard A Raines	1 Mar 11
Dr. Richard A. Raines (Member)	Date
William Kimball	1 Mar 11
Mr. William B. Kimball (Member)	Date

AFIT/GCE/ENG/11-05

Abstract

This thesis addresses the problem of identifying email spear phishing attacks, which are indicative of cyber espionage. Spear phishing consists of targeted emails sent to entice a victim to open a malicious file attachment or click on a malicious link that leads to a compromise of their computer. Current detection methods fail to detect emails of this kind consistently.

The SPEar phishing Attack Detection system (SPEAD) is developed to analyze all incoming emails on a network for the presence of spear phishing attacks. SPEAD analyzes the following file types: Windows Portable Executable and Common Object File Format (PE/COFF), Adobe Reader, and Microsoft Excel, Word, and PowerPoint. SPEAD's malware detection accuracy is compared against five commercially-available email anti-virus solutions. Finally, this research quantifies the time required to perform this detection with email traffic loads emulating an Air Force base network.

Results show that SPEAD outperforms the anti-virus products in PE/COFF malware detection with an overall accuracy of 99.68% and an accuracy of 98.2% where new malware is involved. Additionally, SPEAD is comparable to the anti-virus products when it comes to the detection of new Adobe Reader malware with a rate of 88.79%. Ultimately, SPEAD demonstrates a strong tendency to focus its detection on new malware, which is a rare and desirable trait. Finally, after less than 4 minutes of sustained maximum email throughput, SPEAD's non-optimized configuration exhibits one-hour delays in processing files and links.

Acknowledgements

I am truly grateful for my thesis adviser, Dr. Barry Mullins, for his perfect blend of mentorship, trust, and encouragement. I also thank Bill Kimball for his contagious enthusiasm for brilliant thinking, Tom Dube for his challenging insights and trustworthy advice, and Dr. Rick Raines for fostering the environment that allows men like this to be great at what they do.

I thank the love of my life, who fully supports the pursuit of my passions and journeys alongside me with encouragement. Because of her, I have a contentment that can only come from knowing I'm not alone in my pursuits.

Ultimately, I thank the Lord for giving me the passions that He has. What a wonderful feeling to know my deepest desires are put there by Him for my pleasure and His glory.

David T. Merritt

Table of Contents

List of Figures

List of Tables

I. Introduction

Cyber espionage is responsible for an annual loss of billions of dollars in the United States alone [Rep09] [Eps08]. Because of the low cost of entry into and the anonymity afforded by the Internet, any curious or incentivized person can likely gather important information off private, corporate, or government computer networks [USC08]. Proprietary information from a company's innovative products or research and development often holds a high monetary value. If that information is about national defense assets or national strategy decision-making, then the value is arguably immeasurable.

History has shown that espionage traditionally requires strategically-placed spies or monitoring devices tailored and molded to the environment in which they operate. The same is true in the way spies infiltrate computer networks. Inherently, espionage occurs against a highly targeted victim or group. Some examples of this are the insider amongst senior leaders of an organization [Mes08], the undercover detective within a drug cartel, or the classic secret agent planted in a foreign agency. Cyber espionage is no different in that its success is dependent on how well-tailored it is to its targeted victims.

In recent years, the cyber espionage threat has been widely published and acknowledged by computer security analysts as wells as the mainstream public [Kei10] [Zet10] [Mes08] [Hin08] [GET08] [Rep09] [SAN08]. Historically, the most common method for infiltrating a network is through targeted spear phishing emails with malicious file attachments or web site links [Kei10][Zet10] [SAN08]. The infamous attacks against Google and dozens of other corporations in early 2010, dubbed “Operation Aurora”, used

targeted spear phishing emails in this manner [Jac10]. Both the emails and attachments are products of effective social engineering methods that tailor the content to the recipients of the emails. When an unsuspecting, targeted user opens the attachment or clicks on the link, the cyber spy establishes a foothold on the computer and affected network. The spy can then use his specialized malware to search for interesting data on the victim computer or network and exfiltrate this potentially sensitive data, like source code or intellectual property, from the victim network to a place of his choosing.

Because spear phishing emails are targeted, tailored to their targets, and relatively rare when compared to other email-borne malware infection vectors, current technology does not adequately protect against them. Commodity anti-virus applications generally focus their detection on the malicious software that is already known to exist. The objective of this research is to develop an email-based system that integrates automated malware detection algorithms that provide the capability to detect previously unknown malware. Such a system can be used to analyze files and web site links within emails, specifically looking for and recognizing spear phishing emails that commonly use new malware. This capability significantly limits the primary attack vector for cyber espionage. This system provides an invaluable risk mitigation and information protection tool to a person, corporation, or government aiming to protect their trade secrets, intellectual property, or crown jewels from cyber spies.

This thesis is organized into five chapters. Chapter 2 reviews the relevant literature, and it concludes with a review of the research related to the system designed in this thesis. Chapter 3 describes the methodology used to design and evaluate this system. The results and analysis of the experiments is discussed in detail in Chapter 4. Finally,

Chapter 5 concludes this thesis body with a summary of this work, its contributions, and where this work can be extended for future research.

II. Literature Review

This chapter defines spear phishing and discusses the relevant bodies of research needed to perform spear phishing detection. Section 2.1 introduces and defines spear phishing. Next, Section 2.2 discusses the relevancy of traditional phishing detection research. A thorough review of malware detection technologies is provided in light of static, dynamic, signature-based, and anomaly-based analysis techniques in Section 2.3. Email-borne malicious code detection algorithms are described in Section 2.4. This chapter concludes with a description of the research ideas used for this thesis in the context of the spear phishing attack detection framework.

2.1 Spear Phishing Defined

Spear phishing is a phishing attack targeted at a relatively smaller target set, and it usually uses malicious attachments and web site hyperlinks in the email content. In addition, the email content is highly customized to the target and would probably mean almost nothing to an email recipient outside the target group or organization. Spear phishing emails sent to a handful of selected victims is indicative of cyber espionage. In addition, if the content of the email is very specific and relevant to the industry, then this would be a telltale sign of cyber espionage. The same thought process applies to a compromised web site that hosts information or services that cater to a select business, organization, or niche market.

One way to differentiate between spear phishing and conventional phishing attacks are by the level of sophistication of social engineering require by the attacker. For phishing emails, knowing the demographics, locale, or common financial institutions

in an area aid an attacker in customizing their phishing attack. However, for spear phishing attacks, both the emails and attachments/links are the products of very focused social engineering methods that tailor the content to each individual recipient of the emails.

Spear phishing attacks are email-borne infection vectors, so there is a relatively long and arbitrary amount of time from when an email is delivered until the malicious file or code compromises its target. Also, spear phishing has a different purpose than a worm or email virus. Its goal is not to indiscriminately spread far and wide but to surreptitiously establish a foothold into a specific network or system.

2.2 Detecting Phishing Attacks

Identifying phishing attempts is a difficult and unsolved problem due to the inherent vulnerability residing at the receiving end of phishing emails—a human. The prevalence of phishing web sites and emails attests to the success phishers are having with their attempts. When a web site or email emulates a known legitimate site or email, it is relatively easy to fool most Internet users. While phishing training may help the human only slightly, significant advancements are made toward effective technical solutions that are categorized into two groups: content-based filtering and application-based filtering.

2.2.1 Content-Based Filtering

Content-based filtering refers to statistical analysis, data mining, feature set selection, machine learning, and/or heuristics-based detection mechanisms applied to either email content or web site content.

Fette et al. establish a machine learning algorithm on a feature set designed to highlight human-targeted deception behaviors in email [FST07]. Their approach is named PILFER, and it is a machine learning-based approach to classifying phishing attempts. PILFER uses data directly present in email as well as data collected from external sources. This combined approach creates a feature vector, which is used to train a model for classification. Their feature vector consists of 10 features: Internet Protocol (IP) addresses within web links, age of linked-to domains, non-matching links, "Here" links to a non-modal domain (anomalous links to the non-dominate domain present in the email), HTML (Hyper Text Markup Language) emails, number of links, number of domains, number of dots (e.g., www.this.is.a.bad.site.com), contains JavaScript, and output from third party spam filters. PILFER inputs this feature vector into a random forest as a classifier, where numerous decision tress are created. Preliminary experiments show a 96% detection rate with only a 0.1% false positive rate over 860 phishing and 6,950 non-phishing emails.

L'Huillier et al. propose an online phishing classification scheme using adversarial data mining and signaling games in [LWF09]. They implement a game-theoretic data mining framework that uses dynamic games of incomplete information to build a classifier to detect phishing attempts. The feature set consists of email content-based features, of which there are four categories: email structures related to different email formats, properties of every link in a message, HTML/JavaScript/forms used, and the SpamAssasin's output score for the email in question. This work achieves a high detection accuracy of 99%.

Bergholz et al. propose a number of novel features that are tailored to phishing email detection [BDG+10]. These new features extend the work of L'Hullier et al. by adding a word list to their basic feature set, and advanced graphical features are added as well. These graphical features are image distortion (i.e., attempts to defeat character recognition tools), logo detection (i.e., compared to original logo), and hidden text salting. Hidden text salting consists of random strings, spacing, coloring, spelling, etc. to fool automated appliances but remain invisible to humans. These features are passed into a text classification-based classifier (e.g., random forests or support vector machines). Experiments with these novel features yield a 99.46% accuracy rate, which is slightly higher than that reported by L'Hullier et al.

2.2.2 Application-Based Filtering

Application-based filtering refers to a specific method of implementing a phishing detection or prevention mechanism. This category encompasses email client or web browser plugins as well as modified email architecture.

Zhang et al. developed an automated test bed for evaluating anti-phishing tools in [ZEC+07]. They evaluate 10 popular appliance-based anti-phishing tools using 200 phishing URLs (Uniform Resource Locators, or links) from two sources and over 500 legitimate URLs. The results of their evaluation show that only one of the tools could consistently identify over 90% of phishing URLs. However, this same tool also had a 42% false positive rate. In addition, the authors point out numerous methods to exploit vulnerabilities in multiple anti-phishing tools that resulted in phishing sites being labeled as legitimate. Most of the tools use a blacklist of URLs that they would obtain

dynamically and frequently. Only one tool uses heuristics-based detection instead of an explicit blacklist. This tool also has high false positive rates. The major contribution of this effort is the authors' conclusion that the success of anti-phishing tools using blacklists relies on very large amounts of data being collected frequently.

Crain et al. propose a tool to assist users in identifying legitimate emails [COP10]. This tool, called *Trusted Email*, allows companies to establish keys with their clients/customers. This key is used to sign and encrypt emails between the legitimate company and its user. This approach's strength is that it uses existing technology in a novel way to dramatically improve email security. A client-based plugin provides feedback to users when: 1) a key establishment email arrives, 2) a signed email arrives, and 3) a forged email is detected. A small pilot study shows that all users reject all emails marked as phishing, and they also accept all emails that are signed. However, most of them also rejected all unsigned, legitimate emails, which may be a result of the small group of people and their insight into this research.

2.2.3 Limitations of Phishing Detection

The content filtering techniques focus their detection on anomalous behavior indicative of phishing. This implies that all phishing attempts use non-standard behavior. However, spear phishing specifically emulates a valid user behaving in a legitimate manner and emailing appropriate recipients. Therefore, the content-based filtering algorithms likely will not recognize legitimate-looking spear phishing emails.

On the other hand, application-based filtering shows promise, but it relies heavily on the use of blacklists that must be constantly updated. But there is an inherent

challenge with this: any new phishing attempts will have to be discovered first before it can be added to a known bad blacklist. Even heuristic-based detection suffers from unacceptable false positive rates [ZEC+07]. Therefore, current application-based and content filtering-based phishing detection techniques likely will not catch spear phishing attacks, especially ones crafted and targeted for the purpose of cyber espionage.

2.3 Detecting Malware

Malware comes in many forms with many names: virus, Trojan, worm, downloader, rootkit, keylogger, adware, spyware, and more. For simplicity and convenience, the root of the name (i.e., "malicious software") is used to define the generic use of the term. Any unwanted and malicious program or code running on a system is referred to as malware.

Malware detection is the implementation of a technique or techniques that attempt to identify programs or code that behave in a malicious manner contrary to the intended use of a system. Naturally, detection of unknown malware is the goal, assuming a cyber spy will use sophisticated, novel malicious programs to establish footholds on a computer and within a network. A malware detector typically takes two inputs: 1) the program or code under inspection, and 2) its knowledge of malicious behavior [IdM07]. While there are a myriad of techniques currently in use operationally and academically that reliably detect most malware on a system, these techniques can be categorized into three analysis methods based on how they gather information to detect malware: static, dynamic, or hybrid [IdM07] [HoB05]. These analysis methods are further broken down into signature-based and anomaly-based detection, which differ in their manner of

categorizing their knowledge of malicious behaviors. All of the following methods of malware detection focus on Windows Portable Executables due to the overwhelming prevalence of this type of malware in the wild [VxH10].

2.3.1 Static Analysis Techniques

Static analysis gathers information about malicious behavior using syntactical or structural properties of the program under inspection [IdM07]. It requires establishing sets of file features, or feature vectors, based on file content. This file content typically consists of byte sequences (n-grams), metadata, and/or sequences of instructions and application programming interface (API) or system calls. One advantage of statically analyzing code is that, in general, it can be done relatively quickly without the need to execute the malware. Another advantage is that the program, ideally, can be analyzed holistically due to the availability of all the malware's code. Code obfuscation can make this more difficult, but, more often than not, malware is not obfuscated. Thus, static analysis can yield insight into how the malware is programmed and not just visible behavior at runtime.

2.3.1.1 Signature-Based Static Analysis

Signature-based analysis of malware has historically been a euphemism for commodity anti-virus products. However, the use of the term in this paper simply means that the detection algorithm compares a suspect program against *known malicious features*. Signature-based analysis includes most of the automated malware detection mechanisms. This is because most data mining methods use machine learning on feature sets based on n-grams, strings, instructions/code, file headers, and program structure.

Schultz et al. is an early adopter of a data mining technique for malware analysis. In [SEZS01], they use static analysis-based data mining for detecting new malicious executables. They use three data mining schemes to identify malicious Windows or MS-DOS executables: DLL information, strings, and byte sequences (or n-grams). These schemes differ in their approach to extract feature sets from the executables. These feature sets were used to train RIPPER (an inductive rule learner), Naive-Bayes, and Multi Naïve-Bayes (with voting) classifiers. The approach with the best detection accuracy is the one using the GNU *strings* program to extract strings as feature sets for a Naïve Bayes classification algorithm. However, Schultz et al acknowledge that strings are not robust and can be sufficiently evaded by encrypting the malware.

A similar approach is used by Kolter et al. in [KoM04] and later in [KoM06], where they used data mining and n-gram analysis to tackle two malware detection issues: (1) classifying between benign and malicious executables, and (2) categorizing malicious executables according to their payload. This method uses n-gram analysis to determine the n-grams with the highest information gain. The top n-grams become the feature sets which the classifier algorithms use to determine if an executable is malicious. These n-grams are used to train classifier algorithms based on the following inductive learning methods: Instance-Based Learner, TFIDF, Naïve-Bayes, support vector machines (SVMs), decision trees, boosted Naïve-Bayes, boosted SVMs, and boosted decision trees. The boosted decision tree performs better than the rest of the classifiers. This methodology has been fielded as an application called MECS, the Malicious Executable Classification System. One acknowledged limitation to this method is its high computational overhead when selecting features.

Abou-Assaleh et al. continues this work in the area of byte-sequences and data mining by using a Common N-Gram (CNG) analysis classification method to create malware profile signatures in [ACK+04] and [ACKS04]. These profile signatures are class profiles of normalized frequencies of the most frequently-appearing n-grams. The authors use n-grams between one and 10 bytes in size, and they set lower and upper bounds on the number of n-grams used to 20 and 5,000, respectively. The experiment in [ACK+04] achieves an average accuracy of 98%, but it only tested 65 Windows executables. The experiments in [ACKS04] use almost 800 samples of Windows executables, but the average detection rate dropped to 91%.

Henchiri and Japkowicz use machine learning and knowledge of malware family types for feature selection in [HeJ06]. Specifically, they focus on intra-family and inter-family n-gram thresholds to strategically select or eliminate features for the final feature selection. This focus on malware family types for feature selection is the first of its kind. Experiments with 3,000 samples (approximately half of them malicious) and varying feature sets show a detection accuracy consistently in the mid-90th percentile.

Sung et al. pioneers the use of sequences of Windows API calls in a signature-based methodology in [SXC+04]. The authors create a signature-based detection system called Static Analyzer of Vicious Executables (SAVE) that extracts API sequences from suspect programs and compares these sequences against a signature database of known malicious behavior. Each API call is mapped to a global 32-bit integer identification number. The 16 most significant bits represent the Win32 module, and the 16 least significant bits represent the API call in this module. An API calling sequence is the sequence of these 32-bit numbers. The similarity of the API sequence with that of

signatures in a database of malicious API sequences is determined. The similarity between the API sequences under investigation and known malicious sequences is based on the cosine measure, the extended Jaccard measure, and the Pearson correlation measure for similarities between sequences. If certain sequences are deemed sufficiently similar, then the program is labeled as malicious. On only 20 malware samples, this approach successfully detected all of them. However, an experiment on larger sample sizes is needed to prove the reliability and robustness of this technique. Also, this method of malware detection can be evaded by polymorphic and metamorphic malware.

Ye et al. extend [SXC+04]'s work on API call sequences with their development of an Intelligent Malware Detection System (IMDS) that uses Objective-Oriented Association (OOA) mining-based classification in [YWL+07]. This work requires the development of a Portable Executable parser to construct the API execution sequences. The authors generate rules based on these API sequences by using their own extension to the FP-Growth algorithm, called the OOA_Fast_FP-Growth algorithm. They test their algorithm against nearly 3,000 executables, and IMDS achieves a 93% accuracy rate in detecting malware. However, because the feature set is based on API sequences, this technique is limited by the same polymorphic and metamorphic evasion techniques as the malware detection scheme in [SXC+04].

In [CJS+05], Christodorescu et al. use abstract models, or templates, that describe the behavior of malicious code. These templates of malware signatures consist of a 3-tuple of instructions, variables, and symbolic constants. These templates are formed in an attempt to generalize the signature of an instance of malware while maintaining the semantics of the malicious code's behavior. This algorithm successfully detects all

variants of certain malware, but the sample size is relatively small. Additionally, this scheme relies on IDA Pro's ability to disassemble a binary accurately. Otherwise, the algorithm's detection algorithm is significantly hindered.

In [TSF09], Tabish et al. show that malicious and benign files are inherently different at the byte level. Thus, they use statistical analysis on byte-level content of a file divided into fix-sized blocks. Then, this approach uses statistical and information-theoretic features for these blocks to quantify the file content at the byte level. This scheme is tested against trained classifier models for six common benign file types (DOC, EXE, JPG, MP3, PDF, and ZIP) and six common malware types (backdoor, Trojan, virus, worm, constructor, and miscellaneous). The results of these experiments show a detection accuracy over 90% for all tested malware types. While this scheme also shows a relatively high accuracy for classifying malware into families, its overall detection accuracy appears to be no better than most other malware detection methods.

2.3.1.2 Anomaly-Based Static Analysis

Most static analysis-based malware detection focuses on characterizing the unique aspects of malicious files. On the other hand, anomaly-based analysis focuses on characterizing legitimate files and then looks for anomalous file behavior.

In [SWL07], Stolfo et al. focus on a new type of stealthy malware threat called embedded malware. Their approach uses statistical analysis on byte-level file content to detect anomalous files segments that may be indicative of embedded malicious code. This technique attempts to model numerous benign file types to produce a model revealing what all files of each type should look like. Anomalous and suspicious

behavior is indicated by any deviation from these models. Each file type is represented by a set of statistical n-gram models based on a compact representation of each file type, called a *Fileprint*. This detection scheme is put to experiment using three scenarios: 1) detecting malware embedded in a randomly chosen benign file, 2) distinctly detecting malware amongst benign executables, and 3) identifying obfuscated, self-encrypted files. This technique is able to detect between 72% and 95% of malicious code embedded in PDF files, depending on the location of the embedding. The detection rate for the malware versus benign executable and self-encrypted files varies widely, which suggests that the comparison method may have been too weak for reliable malware or malicious encrypted file detection in general.

In [SKF08], Shafiq et al. enhance the pioneering work of Stolfo et al. by using statistical anomaly detection to identify embedded malware and locate its position within an infected file. This technique addresses the issue of commodity anti-virus software's inability, in general, to detect embedded malware using their signature-based detection engines. This technique characterizes the statistical properties of a benign file using Markov n-grams, which are conditional n-gram distributions (as opposed to traditional n-grams). The authors conclude that a simple n-gram distribution does not yield enough information to accurately identify embedded malware. This algorithm then uses an entropy rate to quantify the variations in the Markov n-grams of a benign file that are caused by embedded malware. The algorithm looks for anomalous entropy rates that do not fall within the Gaussian distribution of benign entropy rates. While this method does require a training phase, malware is not required to train this algorithm’s detector,

making it a "true" anomaly detector that relies completely on a robust model of benign behavior.

This approach does have several limitations. While this method of embedded malware detection outperforms commercial-off-the-shelf (COTS) anti-virus products and the few other embedded malware detectors, it does so at the cost of a high rate of false positives. The false positive rates for .doc and .pdf files are high due to the inherent ubiquity of embedded objects within these file formats. Because of this, the benign behavior model for these file types take into account entropy rates that are hindered by numerous perturbations from embedded objects. Also, this method of detection is vulnerable to a mimicry attack that shapes the embedded malware to have a statistical distribution similar to "normal" or benign behavior.

2.3.1.3 Limitations of Static Analysis

There are numerous limitations to static analysis. Investigating a program's full functionality may never be possible if there are complex inter-component/system interactions between the malware and other collaborative code from other programs, as discussed in [LeM06]. Also, in their work on testing malware detectors, [ChJ04] explain how code/data obfuscation caused by encryption, packing, polymorphism, and metamorphism is a significant obstacle to overcome, especially if the static analysis-based technique is to be automated. COTS anti-virus products are notoriously vulnerable to these evasion techniques. For example, on-access obfuscation, where instructions are decrypted only during execution, makes static analysis extremely difficult and time-consuming [ChJ04].

The efforts of Moser, et al. in [MKK07] further address the limits of static analysis for malware detection. They specifically focus on the viability of evading semantics-aware detection schemes, where the behavior of the malware is modeled abstractly to determine whether a specific piece of code exhibits a specific behavior or function. Moser et al. accomplish this by using a primitive known as an opaque constant, which refers to a code sequence that loads a constant into a register and whose value cannot be determined statically. Opaque constants strategically replace certain register load operations with semantically equivalent instructions, thus generating a code sequence that always produces the same result. Effectively, control flow can become scrambled, and data locations and usage can be hidden from static analysis. This technique is applied to the source code of the target program, which allows much flexibility in applying these obfuscation transformations. In fact, the authors prove that the creation of an algorithm to determine the precise result of an opaque constant-obfuscated code sequence is an NP-hard problem.

2.3.2 Dynamic Analysis Techniques

Detecting malware by analyzing the code during execution is called dynamic analysis. This run-time analysis technique provides relatively immediate and measurable empirical evidence of what an unknown binary is doing or trying to accomplish. As a bonus, dynamic analysis is effective against binaries that obfuscate themselves or are self-modifying. This is due to the fact that the destiny of all programs is to be run on a system, so when the program is running, its behavior and subsequent system modifications can be captured.

2.3.2.1 Signature-Based Dynamic Analysis

Kephart and Arnold pioneered the efforts for automated extraction of malware signatures in [KeA94]. They developed an effective extraction method for malware signatures by allowing viruses to infect large numbers of files. With a plethora of various infected files, they harvest byte sequences in sizes of 12 to 36 bytes. This process yields a myriad of signatures, and the ones with the lowest estimated false positive probabilities are selected for the final signature-based detection engine. This methodology was incorporated into IBM's AntiVirus product during the 1990s. Arnold and Tesauro extended this work into a neural network classifier in [ArT00]. They use n-grams and multiple neural networks in a voting procedure to eliminate false positives and aid in detecting unknown Win32 viruses.

Lee and Mody successfully automate malware detection using runtime behavioral data and machine learning in [LeM06]. Their methodology quantifies a file's runtime behavior into a form of sequenced events. It normalizes this data for canonical-based storage in a database. Their scheme constructs classifiers for machine learning with the stored event sequences as input. They use a technique called Opaque Object to represent this classification data. This specific approach allows objects to represent data in rich syntax and semantics. It also yields a similarity distance between any two objects, which factors into their classification method based on clustering. The authors use a Microsoft-developed distributed system of virtual machine-based "workers" with kernel mode monitor agents running on them. As files under investigation are executed, the monitor agent intercepts and monitors all system calls in kernel mode. The authors acknowledge

the limiting factors of lack of code structural information, environment conditions, and ineffective virtualization as obstacles to accurate malware classification.

Willems et al. takes steps towards automating much of the dynamic analysis required by advanced, in-depth malware analysis in [WHF07]. They use their own tool (now a commercial product) called CWSandbox to monitor all malware system calls and generate a detailed report to simplify a malware analyst's task. They use API hooking and dynamic link library (DLL) injection to run CWSandbox as a rootkit, thus evading detection by sophisticated malware. CWSandbox collects information exposing the malware's behavior. This information consists of file modification/creation, Windows registry changes, DLLs that are loaded, virtual memory footprint, process creation, network connections, and miscellaneous events pertaining to kernel driver or protected storage access attempts. This tool is unique in its ability to bridge the gap between automated, autonomous malware detection and in-depth, human-based analysis. Unfortunately, it is still a relatively slow method of automating malware detection, with an effective throughput of only 500 binaries per day per instance of CWSandbox.

Bailey et al. extend Willem et al.'s efforts in fingerprinting malware behavior using runtime system state changes in [BOA+07]. They execute malware in virtualized environments to perform causal tracing of system objects created during the malware's execution. These system events are exported to a server that builds causal dependency graphs of these events. This aids in validating that the events being caused by the malware are not normal system events. This approach goes beyond the capability of CWSandbox by using classification algorithms to automate detection of malware. The authors implement a tree structure based on a hierarchical clustering algorithm. An

inconsistency measure is calculated for this tree structure to break the tree into meaningful groups or clusters. These clusters serve as models to measure program behaviors against.

Ding et al. implement a behavior-based dynamic heuristic analysis approach to proactively detect unknown malware in [DJB+09]. This approach categorizes behavior features based on Win32 API calls and their specific parameters. An automatic behavior-tracing system is developed to collect the behavior features during runtime. The authors opt for two independent detection models: a statistical detection model and a mixture of expert (MoE) model. The malware behavior features are broken into six classes of malicious behaviors related to files, processes, windows operation, networking, registry settings, and windows services. After comparing the results of malicious and benign executables in the context of these six classes, a more detailed 35-dimension feature vector is defined where each dimension accounts for one kind of behavior. This feature vector is the input to the statistical and MoE models for ultimate classification of malware. After experiments, the statistical model of malware detection achieves a 96% true positive detection rate with a nearly 35% false positive rate. The MoE model achieves a 1% false positive rate, but it can only reach a 75% detection accuracy rate.

Dai et al. takes a different approach to dynamically quantify malware behavior in [DGL09]. The authors focus their work on feature set selection via static and dynamic means. This hybrid method of extracting statistical information is relatively rare and, thus, unique. This comprehensive effort uses multiple data mining approaches: simple heuristics (PE headers and strings), n-grams, static instruction sequences, and dynamic instruction sequences. A separate algorithm is used to obtain instruction sequence blocks

of significance, and the top instruction associations are selected for the feature set. The selected feature sets for each data mining approach are utilized for support vector machine training. The dynamic instruction sequence-based SVM outperformed the other data mining approaches with a malware detection accuracy of over 91%.

2.3.2.2 Anomaly-Based Dynamic Analysis

True anomaly-based dynamic analysis is a rare technique for malware detection. Many methods may claim they are anomaly-based, but for the purpose of this research, anomaly-based implies a quantification of benign behavior, not a quantification of heuristic-based malicious behavior. It is difficult to find published work in this area, which implies that this method of malware detection is still novel, or this method is not a worthwhile pursuit. However, the idea of quantifying benign program execution at runtime should be no less daunting than doing it statically. At least one published effort clearly attempts this task, and another non-published effort successfully achieves this, which is discussed in Section 2.5.

Apap et al. present a host-based intrusion detection system for Windows that focuses specifically on registry accesses in [AHH+02]. The Registry is a worthwhile location to monitor runtime execution of programs. This is because Registry activity tends to be regular over time, with most programs accessing only at startup/shutdown or at specific time intervals. Therefore, anomalous and irregular activity may be relatively easy to identify. Additionally, many malware infections launch programs or change keys that have not been launched or changed since the operating system had first been installed. The authors exploit this beneficial scenario by developing a system called

RAD (Registry Anomaly Detection) to monitor registry accesses in real-time to detect the activity of malware. This work is an extension of a network packet header-based anomaly detector called PHAD (Packet Header Anomaly Detection). The RAD system is divided into three basic components: an audit sensor, a model generator, and an anomaly detector. The sensors log registry activity to a database while the model generator reads this data to determine models of normal registry behavior. Finally, the anomaly detector uses the model as a point of comparison for all registry accesses to determine their potentially malicious intent. After training RAD for two days on benign behavior, the system is put to test. It performs well, achieving a 100% detection rate of malicious activity in certain scenarios. However, there are many false positives in most scenarios, and most of these false positives are caused by legitimate processes that did not run during the two-day training phase. This lack of an exhaustive training phase reveals an inherent limitation to dynamic analysis-based anomaly detection.

2.3.2.3. Limitations of Dynamic Analysis

Just as there are limitations to static analysis that are addressed by dynamic analysis, the opposite is true as well. Dynamic analysis-based malware detection generally requires orders of magnitude more time to perform than static-based techniques. Also, the malware has to run for a long enough duration to capture sufficient malware information, but it also has to be run quickly enough to be scalable. This makes dynamic analysis vulnerable to intentional or inadvertent time-delayed evasion techniques. Specifically, when a program under inspection is executed, the monitoring agent only witnesses a single instance of execution of that program. If malware is

programmed to activate at a certain time, dynamic analysis will almost surely miss this behavior. Bailey et al. draw attention to the fact that anti-VM (virtual machine) evasion techniques exist [BOA+07]. Finally, malware that depends on user input may not reveal its full functionality without manual intervention or an advanced clicking emulator to simulate humanlike mouse clicks.

2.4 Detecting Email-Borne Malicious Code

Detecting malware sent by email can be seen as an extension to the malware detection problem in general. However, there are some advantages to detecting malware within special purpose applications. This is further discussed in the next section. It is important to note that many email-based malware detection solutions do not take advantage of the special purpose application, and they are simply commodity anti-virus products that run the same scans with the same signatures used on client-based systems. While this is convenient and necessary to prevent common, known malware from infiltrating a network via email, it still lacks the ability to detect most new malware and 0-day attacks. Fortunately, there is work in this area that has set the precedent for tracking malicious emails.

Shih et al. propose a method of detecting unknown malicious emails [SCY05]. Their method evaluates the malice of an email attachment by focusing solely on the behavior of the email and not the contents of the attachment itself. This technique evaluates three classifiers against four versions of commodity anti-virus software. The three classifiers it uses are the Naïve Bayes, the Bayesian network, and the decision tree classifiers. Their feature sets consist of 11 email content-based features: mail content

type, mail size, MIME (Multipurpose Internet Mail Extensions) format, attachment, number of attachments, attachment size, script language, subject, carbon copy, and recipient. All three of these classification approaches outperform the four anti-virus products in detecting five variants of known malware. However, targeted and tailored spear phishing emails will likely evade a detection technique that relies on anomalous email behavior.

Schultz et al. makes a significant contribution to this field of email-borne malware detection with their seminal work call MEF (Malicious Email Filter) [SEZB01]. MEF is a tool that detects malicious Windows executables using Procmail and a UNIX mail server. This tool offers three key contributions: 1) detection of known and unknown malware attachments, 2) automatic distributed propagation of detection models, and 3) ability to monitor the propagation of malicious email attachments. This framework uses a Naïve Bayes classifier to detect malware, and this classifier is generated by a data mining algorithm trained over a given set of data. MEF has the ability to detect similar but unknown malware, and its probabilistic classification methods allow the tool to identify borderline executables, meaning they are on the border of the threshold between malicious and benign. These borderline cases provide an opportunity for expert analysts to make a determination. In turn, the detection models can be updated with this valuable new insight, and MEF can update a central server with the updated detection models for distribution to other MEF outposts.

MEF also tracks the propagation of email attachments. It stores a unique identifier (hash) for every email attachment in a database as well as log data pertinent to each email. The logs of malicious attachments are sent to the central server. Therefore,

the central server can use the unique identifier and contextual log data to track the spread of malicious attachments via email. MEF is tested against a signature-based malware detection approach that emulates commodity anti-virus software on mail servers. The data set consists of 4,300 files, 1,000 of which are benign. MEF significantly outperforms the signature-based approach with an accuracy rate of over 97%. The experiments do not evaluate MEF's detection model updating or its malicious attachment propagation tracking. Additionally, the detection of unknown malware is not thoroughly evaluated.

Bhattacharyya et al. extends the work of Schultz et al. with their Malicious Email Tracking (MET) system [BaH02]. MET shifts focus away from the three major capabilities of MEF and towards a dramatically improved malicious email propagation tracking capability. MET maintains a database of statistics about the trajectory of email attachments, and this affords the tool a global perspective on the spread of malicious software via email (assuming a global MET presence). This database of email attachment trajectory data also provides the ability to determine all the points of entry of email-borne malware into a network. Another tangential benefit of this technique is the ability to reduce the spread of self-replicating malware through email.

MET gathers the core statistics for each email attachment, which consists of the prevalence of an attachment and its birth rate. Prevalence is the number of times a MET client observes an attachment, and the birth rate is the average number of copies sent from the same user. MET has built-in heuristics to determine if an attachment is self-propagating, and this information can be communicated to a central MET server for distribution to other MET clients. This is how self-replicating malware can be prevented

after the initial infection. The limitation to this approach is obvious: there will be an initial infection and likely multiple infections before the trajectory data begins to line up with the classifier model. Also, a targeted spear phishing attack is designed to emulate legitimate behavior and appropriate recipients. Thus, a detection technique that tracks email trajectory likely will not recognize anything anomalous in a spear phishing email.

Finally, there is one documented framework for detecting 0-day worms and viruses in email, as proposed by Sidiroglou et al. in [SIK+05]. This framework is an email worm vaccine architecture, and it uses the Registry Anomaly Detection (RAD) mechanism, designed by Apap et al. in [AHH+02], to detect malware in emails. RAD monitors Windows registry accesses in real-time to detect the activity of malware. The authors design the system to use a Simple Mail Transfer Protocol (SMTP) proxy between the Internet and the protected email server. This proxy intercepts all incoming emails, extracts all attachments, and runs the attachments on a virtual machine that uses RAD to detect anomalous behavior. Upon detection of a malicious file and therefore malicious email, the SMTP proxy is notified, and the email message is discarded. Experiments are run with publicly available attacks delivered via email, and this system achieves a 100% detection rate with a false positive rate of 5%. However, this anomaly-based dynamic analysis technique has its drawbacks. Because RAD has to be trained on "normal" behavior, many false positives are caused by legitimate processes that do not run during the training phase of RAD. This lack of an exhaustive training phase reveals an inherent limitation to this method of malware detection in emails. Additionally, these experiments do not appear to use unknown malware, which leave the claim of detecting 0-day worms and viruses untested. Finally, not all malicious code alters the Windows registry. This

limits the detection footprint of RAD, which limits the scope of detectable malware offered by this email worm vaccine architecture.

The email-focused malware detection techniques discussed to this point are either unlikely to detect novel malware in targeted emails due to the difficulty in detecting malware that leaves such a small network footprint (i.e., it does not spread indiscriminately like a worm), or their ability to detect unknown malware remains untested. Also, none of these techniques takes into account the download of malware or the exploitation of web browsers caused by URLs within emails. This is a significant drawback to using any of these techniques to detect spear phishing emails that use malicious URLs.

2.5 Spear Phishing Detection Framework

This section's purpose is to extend the literature review into the introduction of the spear phishing detection framework proposed in this research. Special purpose and general purpose systems are discussed first, followed by the principles behind a malware detection approach that combines static and dynamic analysis. Finally, the two malware detection algorithms selected for this research are introduced and discussed.

This research effort attempts to build a spear phishing detection framework that is implemented at the email server/service level of the network. It uses both dynamic and static analysis techniques to detect the presence of malicious email attachments, especially novel/unknown malware.

There are two important questions about this spear phishing detection framework that need to be answered: 1) Why implement this framework at the server/service level

and not as a client-based application? and 2) Why use a dynamic analysis-based approach when it is notorious for its overhead costs in time and resources? Both of these questions are answered in the following sections.

2.5.1 Special Purpose versus General Purpose Systems

Client workstations are general purpose computers, thus they have to be able to handle multiple types of user input, applications, and functionality. Because of this, the set of all possible actions that a general purpose computer can perform is intractably large. Essentially, their behavior can be unpredictable. In addition, client-based computers have to provide virtually instantaneous feedback and response, or real-time computing, to user inputs. This has become the norm and the expectation from virtually all computer users. Because of this, dynamic-analysis based malware detection has to take into account a myriad of possible files, processes, and network connections being created, modified, and/or terminated. Also, the malware detection product cannot introduce much latency, or else the enterprise-wide adoption of the product will meet many obstacles.

Email servers, on the other hand, are intended to be special purpose computers, though not quite as special purpose as an embedded system (i.e., cell phone, DVD player, electronic gadget, etc.). However, in comparison to client workstations, email server behavior is more predictable, and so is its file, process, and network connection creation, modification, and/or termination. Even though commercial email server software is designed to meet the needs of a broad and diverse customer base, it still operates according to a much more limited baseline behavior than a general purpose system does.

Additionally, latency is not a significant issue. Email delays are acceptable to the point that they are not surprising, and many times they go unnoticed by the email recipients. Therefore, malware detection products have more leeway in time to perform more in-depth analysis on email.

In addition, having a malware detection framework at the network boundary can stop malware at the point of entry into a network instead of waiting to stop it at the client. In most situations, this is a preferred approach over letting the malware reach its target before preventing its execution. Typically, there are many network points of entry, but using email as an attack vector has proven to be effective and reliable for cyber espionage. Also, it is easier to manage/administrate software at the server/service-level vice the client/distributed user-level.

2.5.2 Combining Dynamic and Static Analysis of Malware

In almost all related work for malware detection, authors emphasize the need to augment static analysis with dynamic analysis and vice versa. Hybrid approaches are somewhat rare, but the benefits of combining both approaches are synergistic. While static analysis prevails in speed and exhaustive code analysis, it falters in obfuscated code and embedded malware analysis. While dynamic analysis lacks in analysis speed and time-delayed evasion, it prevails in determining actual code execution sequences and functional behavior. Using a spear phishing detection framework that follows a hybrid approach for malware detection alleviates many of the limitations of static and dynamic analysis.

2.5.3 Using Malware Type Recognition (MaTR) for Static Analysis

A very important piece of a spear phishing prevention framework is its ability to identify malicious attachments to emails. There are precedents for accomplishing this using static analysis-based techniques for malware detection. Specifically, the MEF [SEZB01] and MET [BaH02] tools, in addition to Shih et al.'s work [SCY05], demonstrate that using signature-based static analysis in combination with email services is a viable and effective means to detecting email-borne malware.

Malware Type Recognition (MaTR) is a research initiative that extends the malware detection technologies to include the additional context of malware family types [DRP+10]. At the time of this thesis research, MaTR prototype version 1.00 is the current prototype. This prototype implements MaTR's best-performing pattern recognition technique and feature sets to perform automated static analysis for malware detection and malware type classification. The additional context of malware type provides significant actionable information for network defenders. This situational awareness is vitally important for identifying cyber espionage through spear phishing-borne malware, second only to the tool's fundamental ability to determine malware from non-malware. Also, this context can be used to prioritize more aggressive dynamic analysis efforts.

In the context of this thesis, MaTR is a signature-based static analysis scheme that is comparable to Tabish et al.'s work in [TSF09]. Thus, MaTR is one of the only static analysis-based methods for performing malware type classification as well as having very high detection rates (>99%). It uses program structures and anomaly features of malware that are unique to certain classifier models. This method of static analysis is proven to be very accurate in detecting malware. MaTR uses these feature sets to form classifier

models that are used in a two-stage sequence. The first stage is malware detection, and the second stage is malware classification.

Ultimately, there are three primary reasons to use MaTR as the static analysis-based malware detection engine for the spear phishing detection framework:

1) Malware type classification makes the output of the product actionable.
2) High true positive and true negative detection rates with very low false positive and false negative detection rates compared to all other methods of malware detection.
3) Readily available resources for research associated with MaTR at AFIT.

It is important to note the limitations of MaTR as well. MaTR is prone to the same static analysis limitations in general. However, many of the detection-evading techniques are manifested as feature sets of MaTR, thus making it more resilient to these inherent limitations. Additionally, the MaTR prototype is substantially less accurate in classifying malware into types than it is in detecting malware from non-malware.

2.5.4 Using ESCAPE for Dynamic Analysis

The ESCAPE platform is a true anomaly-based, dynamic analysis technique designed to prevent malicious code from executing [Kim10]. The Air Force currently uses ESCAPE in a web crawling implementation to automate the detection and collection of malicious code that traverses the Internet. The key to its success lies in its ability to whitelist, or sign, executable code that is known to be legitimate and subsequently track and prevent the execution of unknown/unsigned code.

ESCAPE is implemented as a device driver at the kernel level of the Windows XP, Vista, and 7 operating systems. Since a device driver runs at the system level, ESCAPE runs in Ring 0 and can thus implement kernel hooks as necessary. ESCAPE hooks the System Service Dispatch Table (SSDT), which is a kernel structure that contains pointers to addresses for the system services. The SSDT is used to look up the function that handles a given system call [HoB05]. Specifically, ESCAPE hooks the NtAllocateVirtualMemory, NtProtectVirtualMemory, NtMapViewOfSection, NTCreateUserProcess, NtTerminateProcess, and NtSetInformationProcess Windows system functions. In the SSDT, ESCAPE overwrites the pointers to these functions to point to its hook function.

Hooking these functions allows ESCAPE to modify the memory protection characteristics of all pages in memory that are allocated or mapped into [RuS04]. This is due to the memory page protection argument that is passed to all six of these functions when they are called. ESCAPE uses these hooks to force every page in memory to be non-executable, essentially enabling hardware-based Data Execution Prevention for every page in memory.

ESCAPE also hooks the Interrupt Descriptor Table (IDT), which contains addresses to the functions that handle each interrupt, in order to hook the Page Fault Handler [HoB05]. ESCAPE then allows pages in memory to be executed only if it has a valid cryptographic signature. ESCAPE pre-computes the HMAC (Hash-based Message Authentication Code) of every executable section within every file image on the system or application under its protection. This is how ESCAPE whitelists, or signs, executable code. This technique does assume, however, that the initial HMAC computation for all

the executable code is performed on legitimate, known good code. Thus, ESCAPE has the ability to detect the execution of new, unknown, or unverified code caused by an attempt to transfer execution flow to an instruction in a page marked as non-executable. Essentially, ESCAPE uses its kernel access and privileges to protect user-level applications and processes from memory corruption exploits.

An additional feature ESCAPE offers is the ability to use a list of exceptions for unsigned, non-whitelisted code that results from legitimate application functionality. This requires a training period where the protected application is used exhaustively in order to determine if any unsigned code attempts to run legitimately. If so, the signature for this code can be added to an exceptions list, thereby reducing the rate of false positives.

ESCAPE has demonstrated its effectiveness in detecting and preventing malicious code execution resulting from Internet browsing to malicious web sites. Its first real-world use is in a virtual environment that spawns numerous web crawlers to visit known or potentially malicious web sites. As the web crawlers visit sites, ESCAPE detects unknown and potentially malicious code execution caused by web-borne exploit attempts against web browsers. Using this code and memory page whitelisting approach, ESCAPE discovers 0-day exploits as well as known exploits without using signature-based matching algorithms.

Although its proactive malicious code detection capability is desirable, ESCAPE is still vulnerable to a return-to-libc attack due to the inherent nature of the attack using only legitimately executable code. Another potential weakness of ESCAPE's code whitelisting technique is manifested when a page in memory is marked as non-executable

even though there is legitimate executable code present. This occurs in scenarios where applications do not use the Windows Data Execution Prevention (DEP) feature [Dat10]. DEP helps to prevent code execution from data pages. Applications like Adobe Reader or Flash attempt to execute legitimate code from data pages marked as non-executable by ESCAPE. This scenario results in a false positive detection of malicious code if the dynamically-created code is not added to ESCAPE's exceptions list. However, the versions of ESCAPE for Windows Vista and Windows 7 have greatly diminished the false positive rate by preventing the most common DEP bypassing techniques.

Additionally, since ESCAPE's malicious URL detection capability is of interest to a system focused on spear phishing detection, a discussion on ESCAPE's malicious URL detection limitations is prudent. ESCAPE is susceptible to certain web crawler prevention mechanisms by web sites not wishing to be crawled. For example, a web page can appear to serve millions of dummy links to fool crawlers into wasting resources by chasing down each link. This is overcome by setting a link threshold per web page visited by the crawler. Also, a web page can cause infinite recursion or infinitely nested links by pointing links to each other. The crawler can avoid this prevention scheme by setting another threshold on the number of nested links to follow. A third example of an anti-crawling mechanism is a web page causing a seemingly infinite file load time. The web server transmits what appears to be an infinite-sized file, which is defeated by setting a file download timeout on the crawler. Finally, if a malicious URL expects a human user to click on a link to initiate an exploit attempt, then ESCAPE may not be able to detect this URL as malicious. The lack of a user agent to click on active content is a limitation. Additionally, all of these extra thresholds and limitations on the web crawler

effectively limit the ultimate capability of the ESCAPE to access and collect information on as many web pages as possible.

Finally, ESCAPE protects against memory corruption exploits targeting specific application versions on specific operating systems, which are sensitive to the target process's memory address space layout. Memory address space layouts change depending on the operating system version and application version. An exploit that works on one operating system and application version may not work on a different version of the operating system or the application. Because of this, ESCAPE is modified and tuned for each operating system and each application version it protects.

III. Methodology

This chapter presents the methodology to design and evaluate the performance of the SPEar phishing Attack Detection system (SPEAD). Section 3.1 addresses the problem definition, and the goals and hypotheses are introduced in Section 3.2. The approach and experiments to achieve the research goals are outlined in Section 3.3. Section 3.4 provides details on how SPEAD is designed. The System Boundaries and System Under Test (SUT) are defined in Section 3.5. The system services and workload are discussed in Sections 3.6 and 3.7, respectively. Section 3.8 describes the performance metrics to evaluate SPEAD. The system parameters and factors are discussed in Section 3.9 and 3.10, respectively. A detailed explanation of the evaluation technique follows in Section 3.11, and the experimental design is highlighted in Section 3.12. Finally, the chapter is summarized in Section 3.13.

3.1 Problem Definition

Current malware detection technology does not adequately protect organizations or individuals from spear phishing emails that use novel, targeted, and tailored email-borne malware. The most common method of detecting email-borne malware is based on an adaptation of host-based anti-viral techniques that search for specific signatures as well as some heuristics of well-known email malware. However, this static signature-matching technique does not adequately prevent spear phishing attacks, which use novel malware. An automated email analysis framework that reliably detects novel malware will significantly limit a primary attack vector for cyber espionage.

3.2 Goals and Hypotheses

The objective of this thesis is to develop and evaluate a framework to perform automated analysis of emails for previously unknown malware in near real time. The proposed system, SPEAD, identifies email transmissions on a target network, parses the emails for web site URLs (Uniform Resource Locators) and specific file attachments, submits these URLs and files to two malware detection engines (i.e., MaTR and ESCAPE), and stores the malware detection results in a database. Because this research is sponsored by an Air Force entity, design decisions are made, wherever possible, to emulate expected behavior on an Air Force base network.

There are four goals of this research:

1) Construct an email collection and processing system that passively obtains emails, parses them for URLs and specific file attachments, and inserts URL and file metadata into a database for automated malware analysis.
2) Modify MaTR's and ESCAPE's execution environments to: 1) receive source URLs and files from a database for analysis, and 2) update the database with the malware detection results.
3) Collect malware detection metrics from SPEAD and from commercial email anti-virus products for comparison and evaluation of SPEAD's effectiveness.
4) Characterize and evaluate the time required (i.e., latency) to perform this automated analysis of email file attachments and URLs under an approximated Air Force base's email traffic throughput.

It is hypothesized that an email collection system can be constructed that: 1) combines two malware detection algorithms to simultaneously and synergistically

address the other's weaknesses and limitations, and 2) detects novel email-borne malware at a higher rate than commodity anti-virus products. In the context of this thesis, *novel* malware refers to malware or malicious code that is unknown to the general public and cannot be found within public forums, databases, or commercial products. Because email transmissions from client to client generally occur in a timeframe on the order of seconds to minutes, it is also hypothesized that this system will identify the presence of malicious emails within one hour, regardless of the sustained email traffic workload.

3.3 Approach

This section provides a cursory look into the design and evaluation approach for SPEAD and how this approach achieves the four goals of this research. While this section is focused on the high-level design decisions, a more detailed explanation of SPEAD's system design is given in Section 3.4.

The high-level view of SPEAD's overall functionality is illustrated in Figure 1. This figure and its explanation summarize how this approach meets the first two research goals. SPEAD's functionality is broken down into three phases:

1) Email Collection: emails are passively collected (i.e., out of band) from a base LAN.
2) Email Processing: emails are parsed for specific file attachments and all URLs.
3) Malware Detection: files and URLs are submitted to malware detection engines.

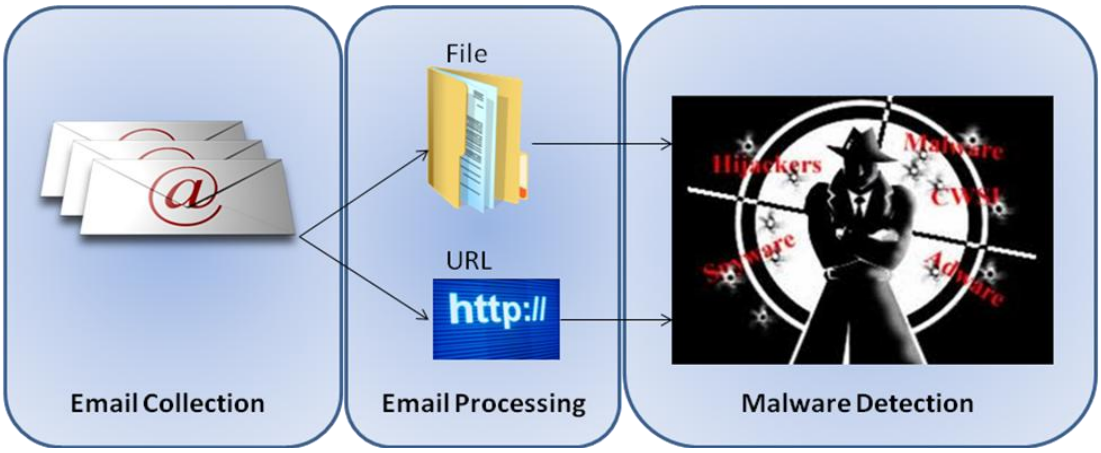


Figure 1: High-level Flowchart of SPEAD Functionality

For email collection, SPEAD collects emails passively as an intentional decision to make this spear phishing detection framework more viable for wide-scale implementation. Within the Air Force and other large organizations, there is a general reluctance to introduce new network infrastructure "inline", meaning the infrastructure becomes another device for network traffic to pass through before reaching its intended destination.

For email processing, SPEAD focuses on the following file types: Windows Portable Executable or Common Object File Format (PE/COFF), Adobe Reader, and Microsoft PowerPoint, Excel, and Word files. These file types are selected for this research because of their prevalence in email spear phishing attacks [Van08].

For malware detection, MaTR analyzes PE/COFF files, ESCAPE analyzes the non-PE/COFF files (i.e., Reader, PowerPoint, Excel, and Word files), and both MaTR and ESCAPE analyze URLs. MaTR is initially designed to perform batch processing of PE/COFF files, and it has been extended to perform malware detection on PE/COFF files downloaded via URLs. ESCAPE is initially implemented to perform a web crawling

mission to detect memory corruption exploits against Internet Explorer, but it has been extended to detect memory corruption exploits against Adobe Reader and Microsoft PowerPoint, Excel, and Word.

The approach for achieving the third research goal involves selecting and installing commodity anti-virus products, obtaining a malicious and non-malicious file and URL corpus, and sending malicious and non-malicious emails to SPEAD and each of the selected anti-virus (A/V) products. Because it is impractical to examine all commercial anti-virus products, five representative commodity anti-virus products are chosen based on the following criteria:

- The product has a history of performing well according to Virus Bulletin's rigorous anti-virus testing [Vir10].
- The product offers a Microsoft Exchange Server component, which represents a common email server for large enterprises, such as the Air Force.
- The product is competitively priced, which increases its viability as a widely scalable solution.

The five selected products are as follows: AVG Internet Security Business Edition, BitDefender Security for Windows Servers, G Data MailSecurity, McAfee GroupShield for Microsoft Exchange, and Microsoft Forefront Protection 2010 for Exchange Server. Each of these five products is installed on a Microsoft Server 2008 R2 operating system running Exchange Server 2007. Because these anti-virus products employ static analysis-based malware detection, the choice of underlying operating system does not affect the product's malware detection accuracy.

Next, a corpus of malicious files and URLs is acquired from various sources, as outlined in Table 1. The malicious PE/COFF, Adobe Reader, and Microsoft PowerPoint, Excel, and Word files are acquired primarily from two sources: the VX Heavens public malware database and two exceptionally large organizations who wish to remain anonymous. Additional Adobe Reader malware is obtained from the Offensive Computing website [Off10] and directly from an industry expert in Adobe Reader malware analysis [Dix10]. The malware collection acquired from the two large organizations is labeled as *Novel* because of the organizations' focus on protecting against cyber espionage and their general reluctance to submit all malware to anti-virus companies. Malware obtained from the public sources are labeled as *Known* to indicate that this malware is more likely to be obtained and analyzed by anti-virus companies.

Table 1: Malicious File and URL Corpus

Sample Type	PE/COFF	Adobe Reader	Microsoft Excel	Microsoft Word	Microsoft PowerPoint	URLs
Known	2,213	112	8	13	7	1,920
Novel	278	91	21	9	4	0
Total	2,491	203	29	22	11	1,920

Malicious URLs are acquired from the online Malware Domain List, which contains almost 60,000 malicious URLs, with the most recent URLs dated 6 January 2011 [Mal11]. Only the URLs submitted since the beginning of December 2010 are used in this research due to the short lifespan of malicious URLs once they are reported. No URLs are collected from private sources, so the entire malicious collection is considered to be known bad.

All malware and malicious URL samples are assumed to be malicious based on the open-source community's vetting of the *Known* malware and URLs as well as the trusted relationship with the two anonymous organizations. Validating every malicious file and URL through malware analysis is outside the scope of this research.

In addition to the malicious corpus of files and URLs, a collection of non-malicious files and URLs, quantified in Table 2, are acquired from a computer security-aware graduate student's computer. The files are reviewed by the student for the purpose of accounting for their existence. Unfamiliar files are discarded to reduce the likelihood of malware making its way into the non-malicious file corpus. The URLs, reviewed in a similar fashion, are obtained from the student's web browser "bookmarks".

Table 2: Non-malicious File and URL Corpus

	PE/COFF	**Adobe Reader**	**Microsoft Excel**	**Microsoft Word**	**Microsoft PowerPoint**	**URLs**
Number of Samples	1,787	382	100	196	170	188

Ultimately, the malicious and non-malicious file and URL corpus are inserted into emails and used in the experiments outlined in the experimental methodology in Figure 2. Experiment 1 determines the optimal configuration for SPEAD and is detailed in Section 3.3.1. The metrics and data collected from Experiments 2 and 3 are used to achieve the third research goal of comparing SPEAD's and the A/V products' detection performance. These two experiments are explained in Sections 3.3.2 and 3.3.3. Finally, Section 3.3.4 describes the experiment to achieve the fourth research goal of characterizing SPEAD's latency.

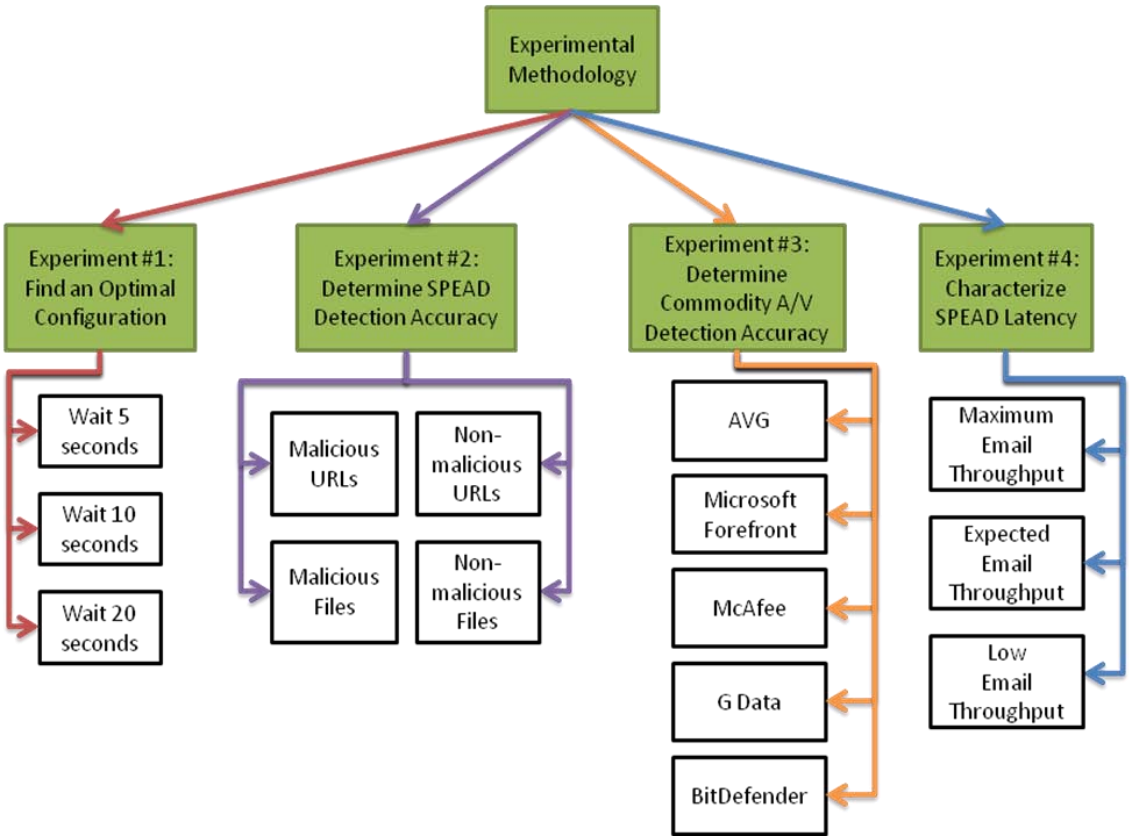


Figure 2: Experiments Used to Achieve Research Goals

3.3.1 Experiment #1: Find an Optimal Configuration

The first experiment seeks to determine the optimal configuration for one of ESCAPE's configuration parameters. ESCAPE's malware detection is based on dynamic analysis, and it must therefore run each file and visit each URL it is analyzing. Chapter 2, Section 2.3.2.3 outlines the limitations of dynamic analysis, one of them being the non-deterministic nature of deciding how long to wait for a process to finish its execution and reveal its true intentions. Because of this, wait times of 5, 10, and 20 seconds are evaluated for malware detection accuracy in three separate tests. A sample set of malicious files and URLs are selected based on preliminary research, where ESCAPE detects their exploitation attempts consistently with a wait time of 30 seconds. This sample set is delivered by email to SPEAD, and the consistency of ESCAPE's detection

accuracy is evaluated for each wait time to determine the optimal configuration for ESCAPE. These three tests are each repeated three times to ensure the metrics are accurate to the 95% confidence level.

3.3.2 Experiment #2: Determine SPEAD Detection Accuracy

The second experiment involves sending malicious and non-malicious emails through SPEAD in order to determine malware detection accuracy metrics. The malicious and non-malicious emails use the entire corpus of malicious and non-malicious files and URLs from Tables 1 and 2. Four tests are performed using varying emails as input: 1) emails with malicious URLs, 2) emails with malicious files, 3) emails with non-malicious URLs, and 4) emails with non-malicious files. These four tests are segregated to simplify the metric collection efforts, and the tests are each repeated three times to ensure the metrics are accurate at a 95% confidence level. The metrics, explained in Section 3.8, are used to evaluate SPEAD's detection performance in comparison with the commodity A/V products, discussed in the next section. Additionally, the first two research goals are validated by demonstrating that all the emails sent into SPEAD are actually received and processed by SPEAD. The list of URLs processed by ESCAPE and MaTR are compared with the list of URLs from both the malicious and non-malicious URL corpus. Similarly, the full collection of files processed by ESCAPE and MaTR are compared with the original corpus of malicious and non-malicious files.

3.3.3 Experiment #3: Determine Commodity A/V Detection Accuracy

The third experiment involves sending malicious and non-malicious emails through each of the five commercial anti-virus products in order to determine their

malware detection accuracy metrics. Just as in Experiment 2, the malicious and non-malicious emails use the entire corpus of malicious and non-malicious files and URLs, and four segregated tests are performed for each A/V product. These four tests are each repeated three times to ensure the metrics are accurate at a 95% confidence level. The metrics are used to evaluate the commodity anti-virus products' detection performance in comparison with SPEAD's, thus achieving the third research goal.

3.3.4 Experiment #4: Characterize SPEAD Latency

The fourth experiment is an abbreviated version of Experiment 2, where a smaller collection of source emails is used. However, the purpose of this experiment is to observe the latency of SPEAD, which is the time SPEAD takes to receive an email and make a determination on the email's malicious intentions. This latency metric is collected for three tests using varying speeds of email traffic sent to SPEAD: 1) a maximum email throughput, 2) an expected email throughput, and 3) a low email throughput. These email throughputs are the speeds at which emails are sent, and they are based on an approximated Air Force base's email traffic throughput. These throughputs are further explained in Section 3.7. The three tests are each repeated three times to ensure latency metrics remain precise within a 95% confidence interval. This experiment seeks to characterize SPEAD's latency in the context of varying email traffic loads, thus achieving the fourth goal of this research.

3.4 System Design

This section delves into the details of the SPEAD system design. It does not cover the details of the underlying hardware and specific software versions, which are

explained in Section 3.11. This section is organized in the context of Figure 1, where SPEAD's overall functionality is illustrated in three phases: email collection, email processing, and malware detection.

3.4.1 Email Collection

SPEAD's email collection flowchart is shown in Figure 3, and it consists of an email server receiving a new incoming email, adding a blind carbon copy recipient to the list of recipients, and delivering the email to the appropriate recipient's mailbox.

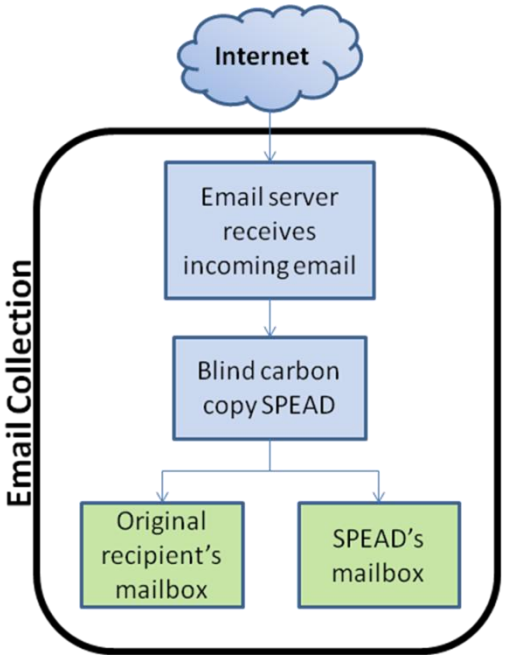


Figure 3: SPEAD's Email Collection Flowchart

The email collection is performed on a Microsoft Exchange 2007 email server running on a Microsoft Server 2008 R2 (Release 2) operating system. The Exchange server is configured with the Hub Transport server role, which is the required configuration to handle all mail flow inside an organization, including email delivery to recipients' mailboxes [Hub06]. A transport rule is created on this email server that is

applied to all emails processed by this email server. This rule adds a recipient to the incoming email by blind carbon copying the email to a specified user's mailbox. The specified user account belongs to SPEAD, and this effectively copies all incoming emails to SPEAD's mailbox.

3.4.2 Email Processing

SPEAD's email processing flowchart is shown in Figure 4, and it consists of downloading emails from a server-side mailbox, parsing the emails for URLs or specific files, and disseminating these URLs, files, and their respective metadata to the malware detection component.

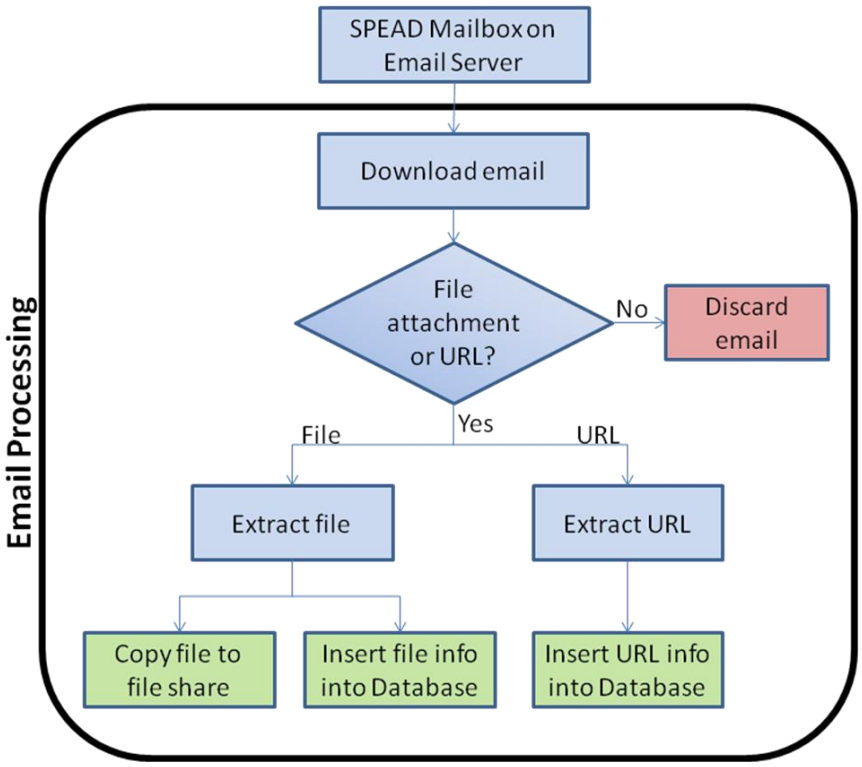


Figure 4: SPEAD's Email Processing Flowchart

The email processing is performed by three Microsoft Windows-based programs: Microsoft Outlook, Visual Basic for Applications (VBA), and Visual Basic scripts

(VBscripts). In the context of Figure 3, Outlook downloads the email from SPEAD's mailbox on the Exchange server, the Outlook VBA module determines if files or URLs exist, and the VBscripts extract the files/URLs, inserts appropriate information into a database, and copies files to a file share. The detailed explanation is as follows:

1) Microsoft Outlook is configured with an Exchange account for its SPEAD mailbox on the Exchange server. As Outlook downloads new emails from the Exchange server, an email rule is run against all new emails as they enter the Inbox, and this rule's purpose is to run an Outlook VBA module to process each email.
2) The Outlook VBA module determines if any attachments or URLs exist within the email. The attachments are easily discovered using VBA's `MailItem` object, which represents an email message in an Inbox folder, and the `Attachments` property, which references all attachments for the specified `MailItem` object [Mal10]. URLs are discovered using a regular expression pattern against which all email content is compared.
3) Upon discovering any file attachments or URLs, the VBA program spawns a VBscript process to handle each file and each URL found in an email. For example, if five emails are received, each with one file attachment, then five VBscript processes are started in parallel to handle each file attachment. The same is true if one email contained five attachments.
4) The purpose of the VBscript process is to insert the appropriate file and URL information into a MySQL database, which serves as a catalyst to start the

malware detection engines. The VBscript behaves according to its input, as follows:

a. If there is a file, the VBscript determines the file type based on the file's binary header information. The script appends the appropriate file extension to the file name and then moves the file to a remote file share server. Finally, the VBscript connects to a remote MySQL database, where it inserts a row containing information pertinent to the file.

b. If there is a URL, the VBscript connects to a remote MySQL database, where it inserts a row containing information pertinent to the URL.

5) The VBscript awaits a response from the malware detection engines for the file or URL in question. The malware detection engines update specific MySQL columns for each file or URL being analyzed. The script tracks this progress by polling the database at regular intervals until the appropriate database columns have been updated for the file or URL row in question. When the malware detection engines' responses are detected, the script performs a final update of its file or URL row before terminating itself. This final update is the latency, or the time it took the malware detection engines to provide an answer. Latency is calculated using VBscript's native *timer* function, which is accurate to the second. A timestamp is taken when an email is received, and another timestamp is taken immediately after a response is received from the malware detection engines. The difference between these timestamps is the latency in seconds.

3.4.3 Malware Detection

SPEAD's malware detection flowchart is shown in Figure 5, and it consists of two malware detection algorithms, MaTR and ESCAPE, polling a database for files and URLs to be analyzed and performing malware analysis on these files and URLs.

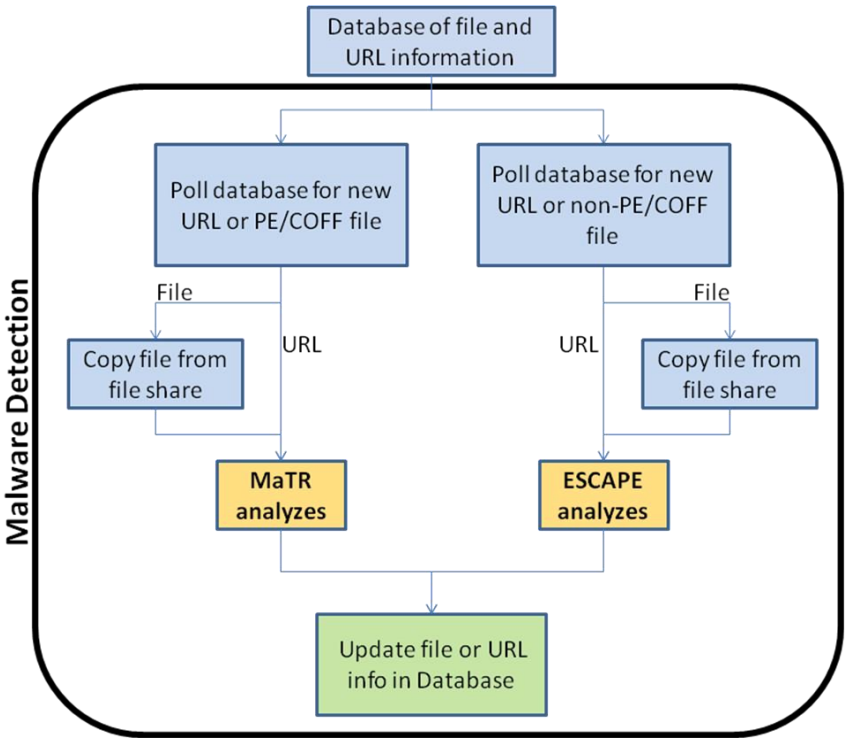


Figure 5: SPEAD's Malware Detection Flowchart

The malware detection is performed by Windows-based MaTR and ESCAPE clients. In the context of Figure 4, the MaTR client polls the database and performs its malware detection algorithm when a PE/COFF file or a URL needs to be analyzed. Similarly, the ESCAPE client polls the database and performs its malware detection algorithm when a non-PE/COFF file or a URL needs to be analyzed. The database is updated with the results of MaTR's and/or ESCAPE's analysis. The details of MaTR's and ESCAPE's roles in malware detection are given in the following sections.

3.4.3.1 MaTR's Role in Malware Detection

The MaTR algorithm is implemented as a command line utility running on a Windows operating system, and it is designed to receive PE/COFF files as input. This Windows client also runs VBscripts to handle the file/URL input from the database as well as the analysis results from MaTR, as follows:

1) Multiple MaTR processes are running in parallel on the Windows client to simultaneously analyze PE/COFF files from email attachments and PE/COFF files downloaded via URLs. The results of analysis are recorded in a local log file.
2) Multiple VBscript processes are running in parallel on the Windows client to simultaneously handle PE/COFF files and URLs from the remote MySQL database. If a PE/COFF file is being analyzed, the file is first copied to the local hard drive from the remote file share. If a URL is being analyzed, a file is downloaded from the URL and its file type is determined. Non-PE/COFF files are ignored. When MaTR finishes its analysis, these scripts read the appropriate log file and update the MySQL database with the malware detection results for the appropriate file or URL.

3.4.3.2 ESCAPE's Role in Malware Detection

ESCAPE is implemented as a 32 bit Windows driver installed on four unique Windows clients, as shown in Table 3. These ESCAPE clients all have Adobe Reader, Internet Explorer, and Microsoft Office installed in order to analyze the four non-

PE/COFF file types. The justification for selecting these specific system parameters is explained in Section 3.9.

Table 3: Four ESCAPE Client Configurations

Windows Version	Adobe Reader Version	Internet Explorer Version	Microsoft Office Version
XP SP2	8.0	7.0	2007
Vista (no SP)	8.0	7.0	2007
Vista SP1	9.0	8.0	2007
Windows 7	9.0	8.0	2007

Furthermore, four copies exist for each of these four configurations. These 16 ESCAPE clients allow for parallel analysis of files and URLs, which is needed to speed up the inherently slow nature of dynamic malware analysis. The number of ESCAPE clients can be extended arbitrarily, but 16 clients are selected to adequately demonstrate scalability of the ESCAPE clients while staying within the limits of the underlying hardware described in Section 3.11. These ESCAPE clients run Python scripts to handle the file/URL input from the database as well as the response from ESCAPE, as follows:

1) Eight ESCAPE clients are running Python scripts to poll the remote MySQL database for non-PE/COFF files to be analyzed. If such a file is ready for analysis, the Python script copies the file from the remote file share to the local disk before opening the file. ESCAPE monitors the execution of the parent application and records its analysis results in a log file. The script reads the log file and updates the MySQL database with the malware detection results for the appropriate file.
2) Eight ESCAPE clients are running Python scripts to poll the remote MySQL database for URLs to be analyzed. If a URL is ready for analysis, the Python

script launches Internet Explorer to visit the web page. ESCAPE monitors the execution of Internet Explorer and records its analysis results in a log file. The script reads the log file and updates the MySQL database with the malware detection results for the appropriate URL.

An additional feature of ESCAPE that is not evaluated in this research effort is its ability to capture a memory dump of the offending process's address space when malicious code is detected. This memory dump is stored on the file share for further analysis as needed.

3.5 System boundaries

The System Under Test (SUT) is the SPEar phishing Attack Detection (SPEAD) system shown in Figure 6. SPEAD consists of the following components: email collection, email processing, file sharing, malware detection database, and the malware detection engine. The Component Under Test (CUT) is the malware detection engine, which consists of two subcomponents: ESCAPE and MaTR.

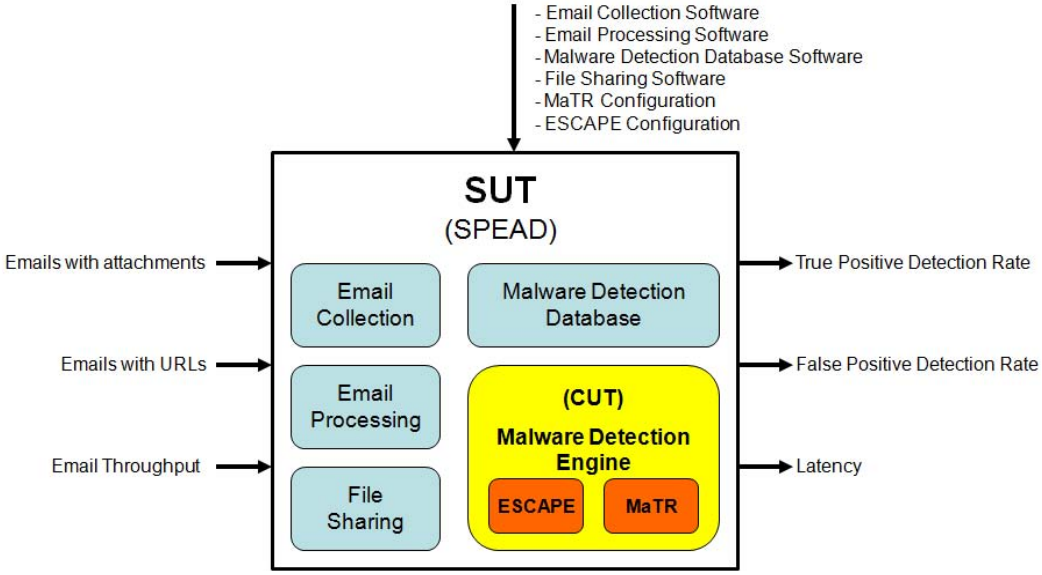


Figure 6: Spear Phishing Attack Detection System

Workload parameters include emails with malicious and non-malicious file attachments, emails with malicious and non-malicious URLs, and the email throughput, which is measured in number of attachments per minute and number of URLs per minute. The workload parameters are discussed in more detail in Section 3.7. The system parameters consist of the software used by the email collection, email processing, file sharing, and malware detection database components. It also includes ESCAPE's and MaTR's configurations. The system parameters are discussed in more detail in Section 3.9. The metrics of the system consist of the true positive malware detection rate, the false positive malware detection rate, and the latency in determining if an email is malicious or not. These metrics are clarified in Section 3.8.

Because Experiment 3 does not use SPEAD, only certain portions of the SUT in Figure 6 apply, such as the workload and the metrics. Specifically, the true positive and false positive detection rates, and not the latency, are collected for Experiment 3.

3.6 System Services

SPEAD provides a malicious email detection service for email servers on local area network gateways to the Internet. The service is successful if the system identifies an email containing a malicious file attachment or URL within an hour to allow for near real time mitigation. SPEAD fails if it incorrectly identifies a malicious email as benign or a benign email as malicious. In addition, failure occurs if SPEAD's latency is greater than one hour.

3.7 Workload

The workload submitted to SPEAD consists of emails sent to an email server that vary in three ways: 1) the presence of a file attachment, 2) the presence of a URL, and 3) the speed at which the emails are sent. Every email used in the four experiments contains exactly one file attachment or one URL within the email body. These files and URLs are pulled from the malicious and non-malicious file and URL corpus.

The speeds at which the emails are sent (i.e., email throughput), are varied only for Experiment 4. Experiments 1, 2, and 3 use the maximum email throughput offered by this experimental setup in order to achieve those experiments with efficient use of time. Email throughput is varied between low, expected, and maximum throughput for Experiment 4 by changing the email sending function on the email sending system. The details of the email sending system are provided in Section 3.11.

The low and expected email throughputs are based on the email statistics from Table 4. The statistics are a summary of a one-week observational study of a large Air Force base's incoming and outgoing emails. The expected email throughput is an estimate of the number of emails with attachments or URLs that SPEAD would be expected to process at a sustained rate at a large Air Force base. This expected throughput is calculated using the one-day maximums over an 8-hour period to approximate peak email usage during a typical work day. The expected email throughput is calculated to be approximately 53 attachments and 1,111 URLs per minute. Because this expected throughput is an estimate, round numbers are used for clarity and to ease the configuration of the email sending system. Thus, this experiment uses an expected email throughput of 60 attachments and 1,000 URLs per minute.

Table 4: Email Statistics from One-week Observation of an Air Force Base

	Attachments	**URLs**
7-Day Total	132,656	2,873,076
1-Day Maximum	25,288	533,363

The low email throughput is based on the overall weekly average, and it is a conservative estimate of the sustained email throughput for non-peak usage. The low email throughput is calculated to be approximately 11 attachments and 249 URLs per minute. Again, more convenient numbers are chosen to aid in configuring the email sending system. This experiment uses a low throughput of 12 attachments and 240 URLs per minute.

The maximum throughput is based on preliminary testing of the email sending system's maximum email throughput. This maximum observed throughput is approximately 232 attachments and 1,422 URLs per minute. This experiment uses an estimated maximum throughput of 300 attachments and 1,500 URLs per minute.

3.8 Performance Metrics

System performance is measured in terms of malicious email detection accuracy rates as well as the latency introduced by SPEAD. The following performance metrics are defined:

True Positive Rate: The percentage of malicious emails identified as *malicious* by SPEAD. This metric is measured by observing SPEAD's response to each malicious email. If every malicious email is identified as such, the true positive rate is 100%. The false negative rate, which is the percentage of malicious emails identified as *legitimate*, is deduced from the true positive rate since the sum of the true positive and false negative

rates is 100%. The true positive rate metric is also measured for the five commodity anti-virus products.

False Positive Rate: The percentage of non-malicious emails identified as *malicious* by SPEAD. This metric is measured by observing SPEAD's response to each non-malicious email. If every non-malicious email is detected as malicious, the false positive rate is 100%. The true negative rate, which is the percentage of non-malicious emails identified as *legitimate*, is deduced from the false positive rate since the sum of the false positive and true negative rates is 100%. The false positive metric is also measured for the five commodity anti-virus products.

Latency: The amount of time it takes an email to propagate through SPEAD. Latency is measured by subtracting the time of the CUT decision determination from the time when SPEAD received the incoming email at the email collector component, as discussed in Section 3.4.2. The latency metric characterizes the maximum and average time delay in SPEAD's determination of an email's malicious intent. This metric determines the success of SPEAD at meeting the secondary experimental goal of providing a near real time response.

3.9 System Parameters

System parameters are the properties of SPEAD which, when changed, affect the performance of the system. Figure 6 lists the system parameters, which are defined as follows:

- *Email Collection Software*: The email collection component of SPEAD uses the Microsoft Exchange 2007 email server running on a Microsoft Server 2008 R2

(Release 2) operating system. Server 2008 R2 is Microsoft's newest server operating system. Exchange 2007 is selected because of its common use in the Air Force. The latest version, Exchange 2010, is considered, but a license cannot be obtained in the timeframe of this research. Regardless, both Exchange 2007 and 2010 have the Hub Transport Server Role, which is used to set up a Transport Rule to blind carbon copy the SPEAD mailbox on the server. Other types of email servers are not considered because of the Air Force-focused nature of this experimental setup. Therefore, this system parameter is fixed.

- *Email Processing Software*: The email processing component of SPEAD uses Microsoft Outlook, VBA programs, and VBscript. Microsoft Outlook is selected as the email client for SPEAD because it is ubiquitous across the Air Force network as the email client of choice, and it is assumed that spear phishing attacks against the Air Force are designed to be successful with Outlook as the email client. VBA and VBscripts are selected because of their inherent interoperability with Windows applications such as Outlook. This system parameter is fixed, but the correctness of the VBA and VBscript code is validated as a part of Experiment 2 because this is original code created for SPEAD.
- *Malware Detection Database Software*: The malware detection database component of SPEAD uses the MySQL database. MySQL is selected because of its proven capabilities for high availability, high performance, scalability, flexibility, and robust transactional support [Top10]. MySQL uses the InnoDB storage engine for transactional support, which allows for unlimited row-level locking [The10]. This

storage engine is crucial for a multi-user concurrency environment such as SPEAD's environment. This system parameter is fixed.

- *File Sharing Software*: The file sharing component of SPEAD uses the Samba service installed on the same Linux platform as SPEAD's malware detection database component. Samba is an open source software suite that provides file sharing services to SMB (Server Message Block) clients [Wha10]. SMB is the service Windows clients use for file sharing. Because the email processing and malware detection engine components of SPEAD use Windows, Samba is selected. This system parameter is fixed.
- *MaTR Configuration*: The CUT uses MaTR as half of its malware detection engine. MaTR's malware detection algorithm uses a decision tree classifier model for malware type classification [DRP+10]. While a different classifier certainly affects the malware detection accuracy, varying this parameter is outside the scope of this thesis. The MaTR prototype is used for this SPEAD prototype. Furthermore, because MaTR analyzes 32 bit Windows Portable Executable and Common Object File Format (PE/COFF) files, it needs a Windows operating system. Thus, the MaTR command line utility runs on the Windows 7 operating system. VBscripts are used as the communication conduit between the MaTR utility and the malware detection database because VBscripts provide convenient functionality with Windows-based applications. Thus, this system parameter is fixed.
- *ESCAPE Configuration*: The CUT uses ESCAPE as half of its malware detection engine. ESCAPE's malware detection algorithm is based on dynamic analysis. Section 3.3.1 describes Experiment 1 and its purpose, which is to establish a

sufficient wait time for a process to finish its execution and reveal its true intentions. Section 4.1 discusses the results of Experiment 1. After Experiment 1, this portion of the ESCAPE Configuration system parameter is fixed and used for Experiments 2 and 4.

Table 3 in Section 3.4.3.2 outlines the four configurations used by the ESCAPE clients. In order to understand why these specific operating system and application versions are selected, it is important to remember that ESCAPE is modified and tuned for each operating system and each application version it protects. Thus, it only detects memory corruption exploits that would have been successful against the operating system and application version ESCAPE is protecting. The selection of the specific operating systems and application versions in Table 3 are a result of a balanced look at two opposing thought processes:

1) When SPEAD operates on a real-world network, the latest operating system and application versions are likely to be employed. To emulate this reality, the latest operating system and application versions must be considered for this research.
2) Because new exploits against the latest operating system (O/S) and application versions are inherently difficult to obtain due to the fact that they are unknown until they are known, older operating systems and application versions must be considered for the sake of experimentation and evaluation of SPEAD's effectiveness in a research environment. To this same end, multiple O/S and application versions must be considered to widen the detection aperture for ESCAPE.

The latter thought process drives more of the reasoning behind the selection of the four specific ESCAPE client configurations. The current Adobe Reader version is not tested due to the difficulty in obtaining exploits against it. Thus, version 8.0 and 9.0 are selected. Table 3 also shows that Microsoft Windows XP, Vista, and 7 are selected because of their current and future use in the Air Force [Ken10]. Internet Explorer 7.0 and 8.0 and Microsoft Office 2007 are selected because of their ubiquitous use on the selected operating systems. All selected operating systems and applications, with the exception of XP and one of the Vista configurations, are unpatched to allow for as many application vulnerabilities as possible for the sake of evaluating SPEAD's implementation of ESCAPE against obtainable malware. Windows XP requires Service Pack 2 (SP2) to run Internet Explorer 7.0, and Vista requires SP1 to run Internet Explorer 8.0.

3.10 Factors

This section describes the factors that are varied during the experiments. Table 5 shows these factors and their associated levels. These factors are selected from the SUT's workload and system parameters. Each of the four experiments uses a portion of these factors and their levels.

Table 5: Factor Levels for the Experiments

Factor	Level 1	Level 2	Level 3
Emails with Attachments	Malicious	Non-malicious	
Emails with URLs	Malicious	Non-malicious	
Email Throughput	Maximum	Expected	Low
ESCAPE Configuration	Wait 5 sec	Wait 10 sec	Wait 20 sec

Experiment 1 focuses on ESCAPE's configuration. Thus, all factors are controlled at the Level 1 factor level with the exception of the ESCAPE Configuration system parameter, where all three factor levels are used. Malicious attachments and URLs are used at a maximum email throughput to measure ESCAPE's malware detection configuration in a time-efficient manner.

Experiments 2 and 3 focus on SPEAD's and the commercial anti-virus products' malware detection metrics. Thus, emails with attachments and emails with URLs are varied according to their factor levels. The email throughput factor is controlled at the maximum level to complete the experiments in a time-efficient manner. Experiment 2 also uses a controlled ESCAPE Configuration factor level that is determined from Experiment 1. Experiment 3 does not use the ESCAPE Configuration factor because SPEAD is not a part of this experiment.

Experiment 4 focuses on the latency metric of SPEAD across varying email throughputs. A sample subset of malicious and non-malicious files and URLs is randomly selected for each of the three email throughput latency tests. The selection of this sample set is clarified in Section 4.3. Thus, only the email throughput factor is varied. This experiment uses a controlled ESCAPE Configuration factor level that is determined from Experiment 1.

3.11 Evaluation Technique

The experiments use a direct measurement-based evaluation technique to determine SPEAD's performance by recording measurements while the system is operating. There are many reasons to pursue this evaluation technique over simulation-

based or analytic modeling-based techniques. Namely, the MaTR and ESCAPE frameworks are already established and can be integrated into this new system. Also, all network activity takes place in a controlled laboratory environment. Because the environment is controlled, the accuracy afforded by this measurement-based evaluation is high and more realistic than a simulation-based or analytic modeling-based evaluation. Lastly, the higher cost typically associated with measurement-based evaluations is fulfilled by sponsor funding for this research effort.

The experimental setup is illustrated in Figure 7. It shows that the experiments are initiated from an email sender system. The email sending system crafts and sends one email at a time to an arbitrary email account on the Microsoft Exchange server. Since this email server is the email collection component for SPEAD, all emails are copied to SPEAD's mailbox on the server. This is required for Experiments 1, 2, and 4.

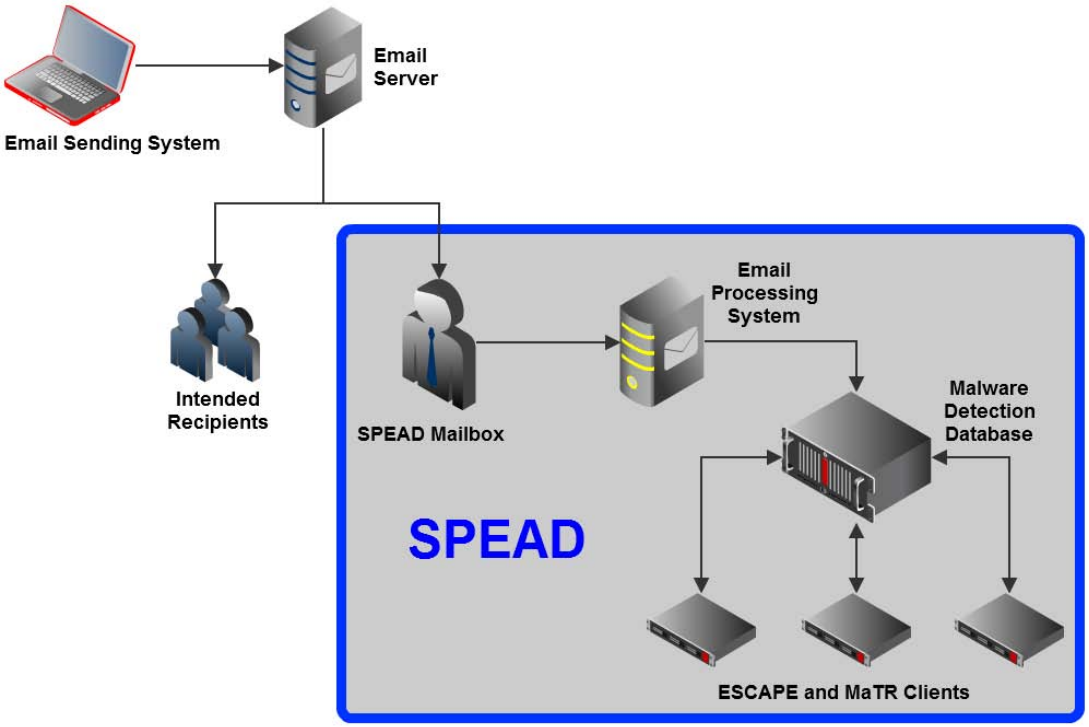


Figure 7: Experimental Setup

The email server also doubles as the server upon which the commercial anti-virus products are installed for Experiment 3. Virtual machine snapshots are used to differentiate between operating configurations for this email server, and additional details are given in the rest of this section.

The experiment environment consists of a collection of hardware and software dedicated to this research effort. The experimental setup consists of the following components:

- Two Dell PowerEdge R610 servers with 64GB of memory, dual six-core Intel Xeon processors at 2.93GHz, four Ethernet ports, and 1TB of storage each. These servers run all of SPEAD's software for the experiments via virtual machines running on top of the ESX 4.1 operating system. These servers are referenced below as ESX Server 1 and ESX Server 2 for clarity.
- One email sending system running on an Ubuntu Linux 9.10 (64 bit) virtual machine image on ESX Server 1. This image has Postfix version 2.6.5 installed, which is an open source email server package for Linux. Mutt version 1.5.20 is installed as the email client, which is responsible for sending the email workload for the experiments. This image is configured with four virtual processors, 16GB of RAM, and one virtual Ethernet adapter.
- One email collection system running on Microsoft Server 2008 R2 with Exchange Server 2007 SP1 loaded as a virtual machine image on ESX Server 1. This image has six snapshots. One snapshot is saved as the baseline email server that SPEAD uses for its email collection component. Five snapshots are saved for each of the following five installed anti-virus products: AVG Internet Security Business Edition

9.0, BitDefender Security for Windows Servers version 3.0, G Data MailSecurity, McAfee GroupShield version 7.0.1 for Microsoft Exchange, and Microsoft Forefront Protection 2010 for Exchange Server. This image is configured with two virtual processors, 4GB of RAM, and one virtual Ethernet adapter.

- One email processing system running on a Windows 7 (64 bit) virtual machine image on ESX Server 1. This image has Microsoft Office 2007 installed, the MySQL Connector/ODBC 5.1.8 driver (Open Database Connectivity), and the following open source command line utilities: Fourmilab's MD5 version 2.2 program and the GnuWin32 File version 5.03 program. This system has the default references for the Outlook Visual Basic for Applications (VBA) editor in addition to the following: ActiveX Data Objects 6.0 Library and VBScript Regular Expressions 5.5. This image is configured with four virtual processors, 16GB of RAM, and one virtual Ethernet adapter.
- One malware detection database running on an Ubuntu Linux Server 10.04.1 (64 bit) virtual machine image on ESX Server 1. This image has MySQL 5.1.41 and Samba 3.4.7 services installed. This image is configured with four virtual processors, 30GB of RAM, and one virtual Ethernet adapter.
- 16 ESCAPE clients running various versions of Windows virtual machine images on ESX Server 2. These images are the four clones of each of the four ESCAPE configurations outlined in Table 3. The XP images are configured with one processor and 1GB of RAM while the others have 2GB of RAM. All 16 images have PyWin32 version 2.6 installed, which is a Python Win32 extension for Windows. Each image has one virtual Ethernet adapter connected to the Internet.

- One MaTR client running on a Windows 7 (64 bit) virtual machine image on ESX Server 2. This image uses two virtual processors, 4GB of RAM, and one virtual Ethernet adapter.

3.12 Experimental Design

The four experiments each use a partial-factorial design with factors and levels selected from Table 5. There are a total of 90 tests required to accomplish all four experiments. These tests are calculated as follows:

- Experiment 1→ 3 ESCAPE configurations * 3 repetitions = 9
- Experiment 2→ 2 email types * 2 levels * 3 repetitions = 12
- Experiment 3→ 2 email types * 2 levels * 5 A/V products * 3 repetitions = 60
- Experiment 4→ 3 email throughput levels * 3 repetitions = 9

Each experiment is replicated three times to confirm the resulting metrics at a 95% confidence level.

3.13 Methodology Summary

This chapter discusses the methodology used to evaluate the performance of an email spear phishing detection system. The four goals of this research or introduced, and the approach and experiments to accomplish these goals are described. The system's design, boundaries, services, workload, and parameters are provided in detail. Performance is evaluated using a measurement-based technique and is based on three performance metrics: true positive rate, false negative rate, and latency. A partial-factorial experimental design is replicated three times for each experiment for a total of

90 tests. Analysis of the results of these tests is used to evaluate the impact of varying the workload and system parameters on overall system performance.

IV. Results and Analysis

This chapter presents and analyzes the experimental results from the four experiments. First, the results of Experiment 1 are explained in Section 4.1. Next, Section 4.2 provides an analysis of the metrics collected from Experiments 2 and 3 to characterize SPEAD's effectiveness. Section 4.3 quantifies SPEAD's latency performance using Experiment 4 results. Finally, the chapter is concluded and summarized in Section 4.4.

4.1 Results and Analysis of Experiment 1

This experiment's purpose is to vary ESCAPE's configuration across three tests to determine which configuration is optimal for accurate malware detection and speed of analysis in terms of only the wait time factor. The time ESCAPE waits for a process to execute or a web site to load is varied between 5, 10, and 20 seconds. A sample set of 15 malicious files and 10 URLs is selected based on preliminary tests, which show consistent ESCAPE responses for these files and URLs. These files and URLs are sent to SPEAD from the email sending system at the maximum throughput. It is important to note that many malicious web sites do not attempt to exploit the same IP (Internet Protocol) address multiple times in a short timeframe to keep automated malware collection devices from collecting their malware. Because of this possibility, this test is repeated two more times on different days to reduce the likelihood of skewed results.

The file results of this experiment are shown in Table 6. The analysis for ESCAPE's file wait time configuration is straightforward. With the exception of the 5-second wait time in Test 1, all other tests and configurations correctly detect all 15 files

as malicious. Because the 20-second wait time offers no apparent benefit, the 10-second wait time is chosen for ESCAPE's file wait time for SPEAD.

Table 6: Results of File Tests for ESCAPE Wait Times

	Files Detected as Malicious		
Wait Time	Test 1	Test 2	Test 3
5 sec	12	15	15
10 sec	15	15	15
20 sec	15	15	15

The URL results for this experiment are illustrated in Figure 8. The 10-second wait time shows the outright best or tied for best performance in all three tests, with a maximum of seven out of the ten malicious URLs detected correctly. It is noteworthy that the 5-second wait time detected one more malicious URL than the 20-second wait time in Test 1, which is counterintuitive. This may be a manifestation of a few of the web sites withholding their exploits after seeing the same IP address twice in a short period.

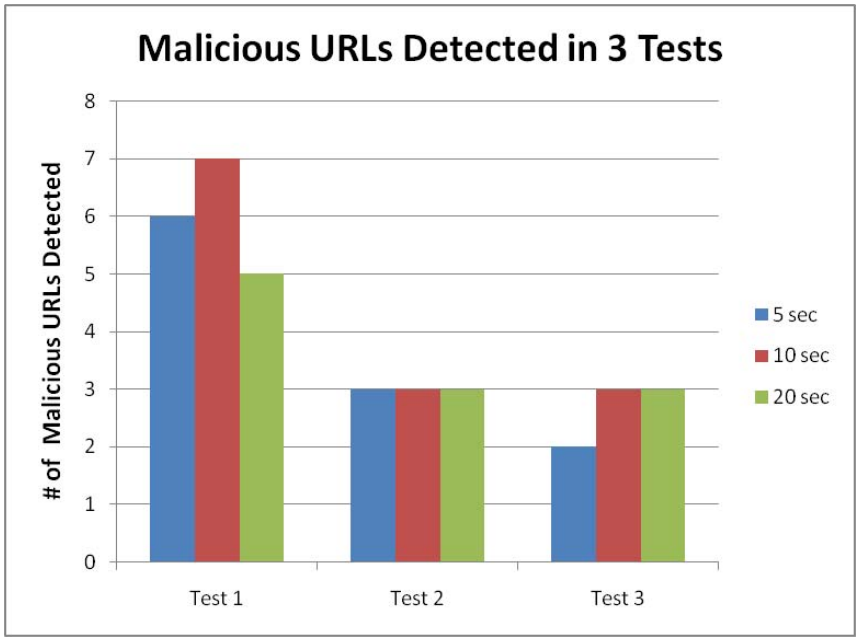


Figure 8: Results of URL Tests for ESCAPE Wait Times

Because of the results of Experiment 1, ESCAPE's wait time for files and URLs is configured for 10 seconds. This SPEAD system parameter is fixed for the remaining experiments, and it represents an optimal wait time configuration only. Other fundamental aspects of ESCAPE's execution environment are not evaluated for optimal configuration, and this is the subject of future work.

4.2 Results and Analysis of Experiments 2 and 3

This section first discusses the validation of SPEAD's design. Then, the file-based detection metrics are analyzed. This section is concluded with an analysis of the URL detection metrics.

4.2.1 Validation of SPEAD's Design

As discussed in Section 3.3.2, the files and URLs processed by MaTR and ESCAPE during Experiment 2 are compared with the original files and URLs from the corpus. Figure 9 shows a collage of screenshots illustrating the validation technique used for the files. SPEAD stores all of these files on the file share on the malware detection database for processing by ESCAPE and MaTR. The Linux program *md5deep* is used to calculate the MD5 (Message-Digest Algorithm 5) hashes of the file corpus on the email sending system. These unique hashes are compared to the MD5 hashes of all the analyzed files stored on the file share. Figure 9 also shows two red underlined Linux commands. The `-x <comparison file>` option for *md5deep* tells the program to compare every newly calculated hash to the list of hashes in the comparison file and output any non-matches. This command results in no non-matches. Conversely, the `-m <comparison file>` option tells *md5deep* to output all matches, of which there are

exactly 5,391. This is the total number of files in the file corpus. The red rectangles in Figure 9 verify that all 5,391 files are processed from the email sending system to the email processing system and, ultimately, to the malware detection database.

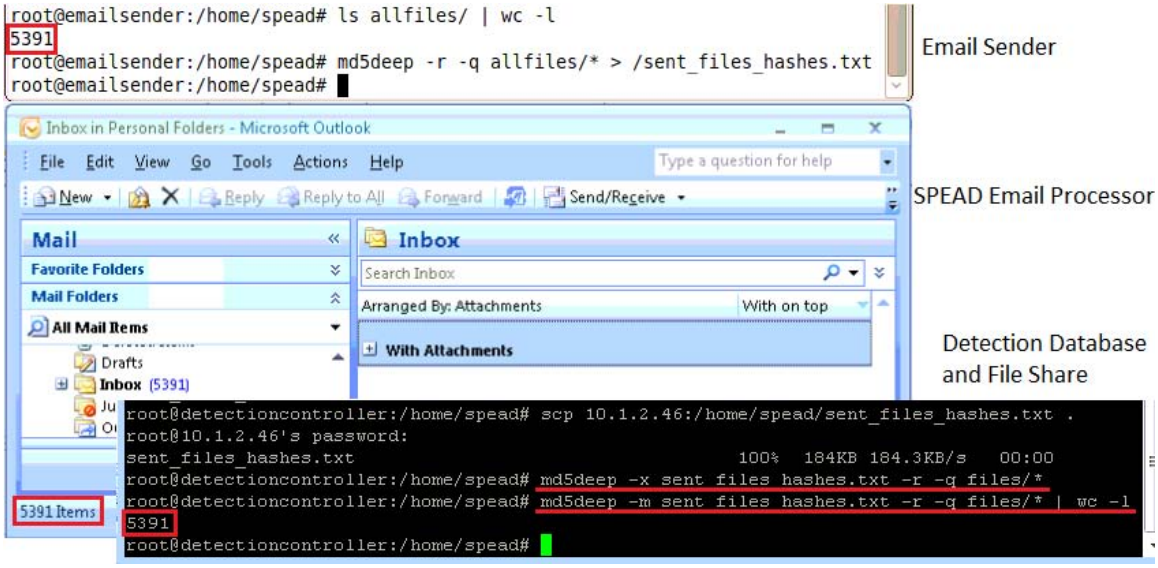


Figure 9: Collage of Screenshots Showing SPEAD File Processing Validation

This MD5 calculation and comparison is made after each of the three iterations of Experiment 2, and the MD5 hash values for the files match every time. This is conclusive evidence that SPEAD correctly recognizes and parses file attachments from emails, validating this portion of the original code written for the email processing component of SPEAD.

The URLs are validated in a similar fashion, but Figure 10 shows a URL that SPEAD processed that it is not intended to analyze. The MD5 hash calculated for the list of all malicious and non-malicious URLs from the URL corpus do not match the hash of the list of URLs in the database. Thus, the application *Notepad++*, a source code editor and *Notepad* replacement, is used to perform a visual comparison of the list of URLs sent and the list in the database. There is only one discrepancy, shown in Figure 10 with the

red minus sign next to it. This URL is from the collection of clean URLs, which has been exported into a file from a graduate student's web browser bookmarks. The email sending system uses this file of exported bookmarks in its original format (i.e., URLs are tagged with metadata). Because of this, the following bookmark is processed:

```
<DT><A HREF="[intended URL]" ADD_DATE="1292535624"
   ICON=" <snipped> Ff"
    >About.com: http://www.state.ma.us/dor</A>
```

The yellow highlight shows a text description for this bookmark that contains a URL within it.

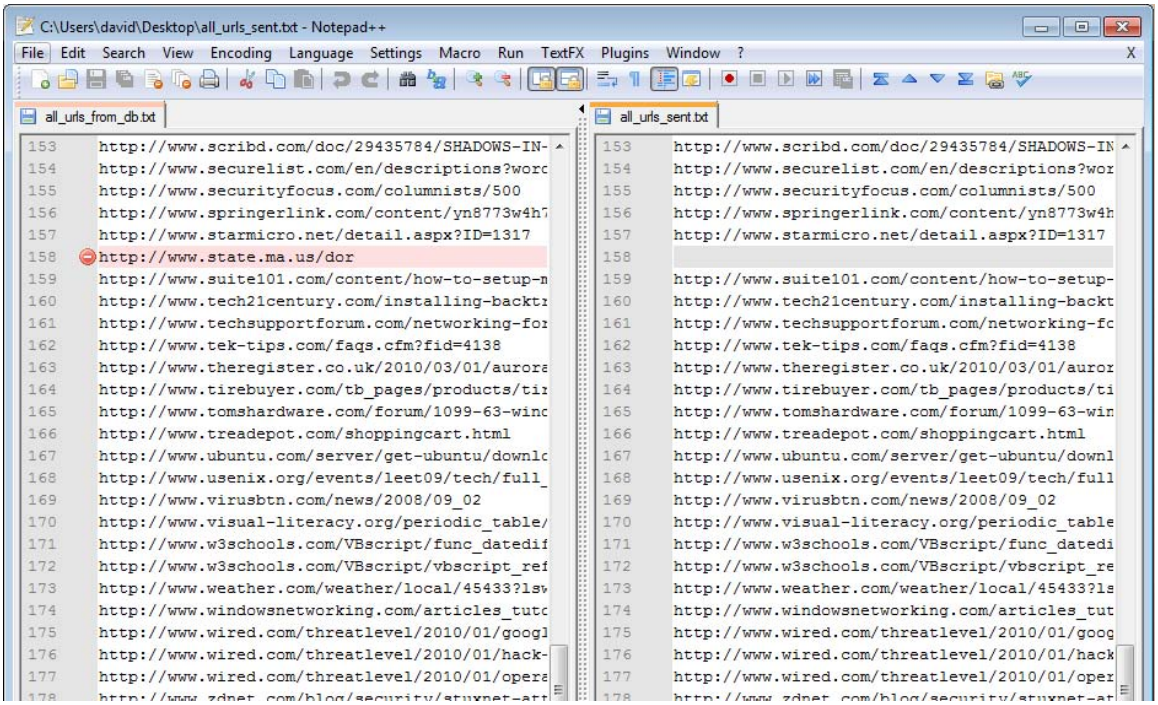


Figure 10: Screenshot of URL Mismatch

This URL is removed from the list in order to perform another MD5 hash comparison, shown in Figure 11. The hashes for the two lists match, which validate that only one URL is mismatched. This URL mismatch appears to be inconsequential considering it is SPEAD's purpose to recognize web site links within emails. Further

testing with real-world emails may reveal flaws in SPEAD's URL processing algorithm, if any. This is discussed as an area for future research in Section 5.3.

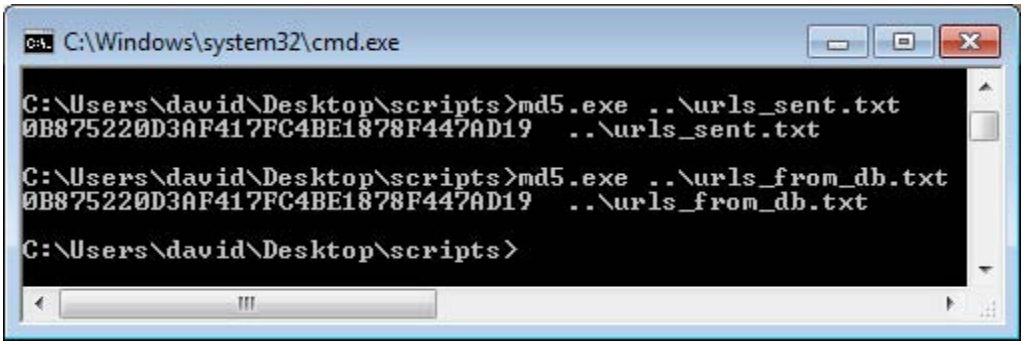


Figure 11: Screenshot Showing SPEAD's URL Processing Validation

The results of this experiment provide convincing evidence that SPEAD correctly recognizes and parses file attachments and URLs from emails, validating the original code written for the email processing component of SPEAD.

4.2.2 Comparing File Detection Metrics

This section contains the true and false positive detection metrics and the analysis of these metrics for SPEAD and the five commodity anti-virus (A/V) products. The overall file detection metrics for SPEAD and the five A/V products are shown in Table 7. These metrics are the mean detection accuracies across three tests for each platform. The green highlights indicate which system had the highest true positive detection rate for each file type. The yellow highlights indicate which systems detected false positives, meaning the systems label a file as malicious when it is not malicious. Section 4.2.2.1 discusses the PE/COFF metrics, followed by Sections 4.2.2.2 – 4.2.2.5, which analyze the non-PE/COFF metrics.

Table 7: Comparison of Overall Detection Metrics for Files

System	PE/COFF		Reader		Excel		Word		PowerPoint	
	True + Rate (%)	*False +* Rate (%)	*True +* Rate (%)	*False +* Rate (%)	*True +* Rate (%)	*False +* Rate (%)	*True +* Rate (%)	*False +* Rate (%)	*True +* Rate (%)	*False +* Rate (%)
SPEAD	99.68	0.39	68.35	0.96	67.82	0.67	13.64	0.00	0.00	0.00
Forefront	95.06	0.36	87.19	0.00	100.00	1.00	95.45	0.00	81.82	0.00
G Data	94.82	0.00	75.37	0.00	51.72	0.00	81.82	0.00	81.82	0.00
BitDefender	93.06	0.00	84.73	0.00	17.24	0.00	31.82	0.00	81.82	0.00
McAfee	90.29	0.00	42.89	0.00	79.31	0.00	68.18	0.00	54.55	0.00
AVG	93.59	0.00	65.02	0.00	20.69	0.00	45.45	0.00	54.55	0.00

The detection metrics are further organized into 2x2 tables of counts of total malware detected and not detected in each test, as shown in the example in Table 8. This allows for Fisher's Exact Test (FET) to be used to calculate p-values for the difference in sample proportions. FET is the "gold standard" of testing tools for 2x2 tables because of its calculation of a p-value that requires no approximation [RaS02]. Furthermore, FET is appropriate for any sample size and for the test of equal population odds. Both of these facts are relevant to the analysis in this chapter due to the relatively small sample sizes of Microsoft Excel, PowerPoint, and Word malware as well as the use of the odds ratio to compare novel and known malware detection rates. The *R* statistical application [Rpr11] is used to calculate the FET p-values for this thesis chapter.

Table 8: Example 2x2 Table for SPEAD Detection Metrics

SPEAD Detecting PE/COFF Malware		
Malware	**Detected**	**Not**
Known	2,210	3
Novel	273	5

The p-value of any particular test is the measure of the credibility of the null hypothesis, which is the hypothesis that the means of the data being tested are equal. If

two data sets are truly equal, the null hypothesis is confirmed. The p-value is the probability that random sampling of the data population distribution could achieve the same result reported in the test. Thus, Figure 12 is a guide for interpreting p-values. A very small p-value, such as 0.005, is usually an indicator that there is strong evidence for the null hypothesis being incorrect. This means the probability that random sampling could achieve the same result is very small, and it is convincing evidence the test result is not due to chance (i.e., the two data sets' means are different). FET uses a two-sided p-value, which allows for the difference in means to be positive or negative. Thus, all p-values reported in this chapter for tests of equal population odds are two-sided.

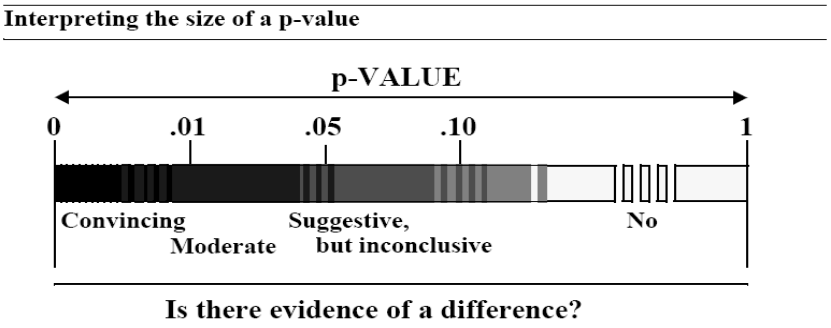


Figure 12: p-Value Interpretation Scale [RaS02]

4.2.2.1 PE/COFF Malware Detection Results

As shown in Table 7, SPEAD's PE/COFF true positive detection rate is clearly the highest, thanks to MaTR's remarkable PE/COFF malware detection algorithm. Because Table 7 reports the mean detection accuracies across three tests, it is important to determine if SPEAD's PE/COFF detection accuracy is truly different from the other platforms'. Tukey's HSD (Honest Significant Difference) Method is used to test all possible pairwise differences in means to determine if at least one of these differences is significantly different from zero at a 95% confidence level. Figure 13 illustrates this

comparison. The six systems are labeled *1* through *6*, with SPEAD being *1*. Visual inspection, highlighted by the red box, shows that SPEAD's mean detection accuracy compared to the others' is significantly different from zero (i.e., dashed vertical line). The p-values for each of SPEAD's comparisons is too small to be reported by *R*, which is conclusive evidence that SPEAD's higher-performing PE/COFF true positive detection accuracy is not the same (i.e., it is different) at a statistically significant level.

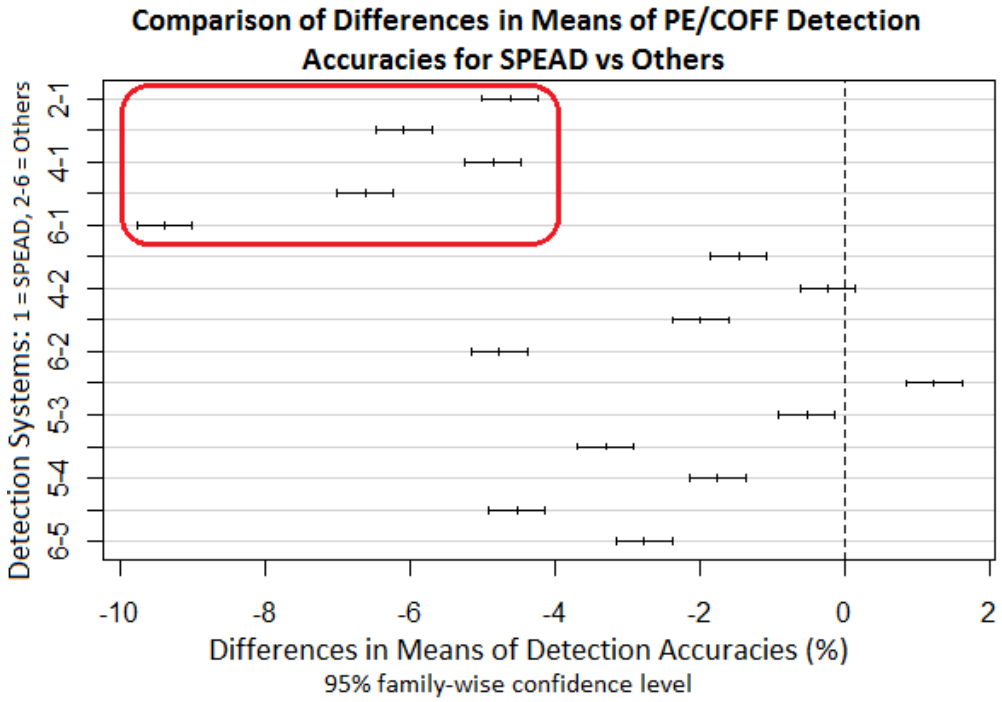


Figure 13: Comparison of Differences in Means of PE/COFF Detection Accuracies for SPEAD versus Others

In terms of the false positive metric, most of the systems do not have any trouble correctly labeling the non-malicious files across all files types in the corpus. Only SPEAD and Forefront record any false positives. This is understandable for malware detection algorithms that attain significantly high true positive rates. It appears that SPEAD's and Forefront's detection algorithms seek to encompass more generic

heuristics of malicious PE/COFF files, as indicated by their achieving the highest true positive rates for PE/COFF files.

Naturally, there is value in quantifying the significance of the difference between SPEAD's and the others' true positive detection rates. Thus, a comparison of the statistical odds of detection between SPEAD and the next best performer, Forefront, reveals that SPEAD's performance may be preferable. The odds for SPEAD correctly detecting a malicious PE/COFF file is 310.4:1. This is calculated by dividing the number of malicious PE/COFF files not detected (8) by the number correctly detected (2,483). Forefront's odds for correctly detecting a malicious PE/COFF file are 19.3:1. Thus, the odds of SPEAD correctly detecting a malicious PE/COFF file are 16.1 times as large as the odds for Forefront correctly making the same determination, with a 95% confidence interval of 7.87 to 33.03. Also, Fisher's Exact Test (FET) for the difference in sample proportions results in a two-sided p-value less than 2.2e-16, providing strong evidence that SPEAD's and Forefront's true positive detection proportions are different (i.e., the ratio is not 1:1) at a statistically significant level.

The same odds ratio comparison is used to determine that SPEAD is 1.2 times more likely to detect a non-malicious file as malicious when compared to Forefront, with a confidence interval of 0.39 to 3.48 and a FET p-value of 1. This very high p-value indicates a high probability that the difference between SPEAD's and Forefront's false positive rates can be attributable to chance. Therefore, the evidence strongly supports the notion that SPEAD's overall PE/COFF detection performance may be preferable when compared to Forefront's when a similar malware population is evaluated.

Even though SPEAD performs PE/COFF malware detection well, the overall detection metrics from Table 7 do not bode well for SPEAD's direct comparison against the five commodity A/V products. A cursory visual inspection of these metrics show that SPEAD ranks fourth, third, sixth, and sixth out of the six systems in terms of true positive rates, respectively, for Adobe Reader, Microsoft Excel, Word, and PowerPoint files. This cursory analysis leads to the conclusion that SPEAD may underperform if it is used as an anti-virus replacement for email servers.

However, the focus of this research is on spear phishing detection, which greatly depends on the system's ability to detect novel malware. Because of this research focus, additional analysis is needed in terms of novel malware detection versus known malware detection.

Table 9 summarizes the PE/COFF malware detection metrics in the context of novel and known malware. As discussed in Chapter 3, Section 3.2, the term *novel* malware refers to malware or malicious code that is unknown to the general public and cannot be found within public forums, databases, or commercial products. The malicious file corpus has been segregated according to Table 1.

Table 9: PE/COFF Detection Results for Novel and Known Malware

System	PE/COFF		
	Novel Detection Accuracy (%)	*Known* Detection Accuracy (%)	False Positive Rate (%)
SPEAD	98.20	99.86	0.39
Forefront	56.47	99.91	0.36
G Data	53.96	99.95	0.00
BitDefender	42.09	99.46	0.00
McAfee	27.70	98.15	0.00
AVG	46.04	99.56	0.00

The odds of SPEAD detecting *novel* PE/COFF malware is 42.1 times as large as the odds of Forefront, the next best performer, detecting the same. This is with a 95% confidence interval of 16.84 to 105.14 and a FET two-sided p-value less than 2.2e-16, which is conclusive evidence to support this large discrepancy in odds. The relatively wide range of the confidence interval for the odds ratio is the result of a large standard error. The standard error for an odds ratio is calculated by taking the square root of the sum of the reciprocals of the four cell counts in a 2x2 table [RaS02], like that shown in Table 8. SPEAD detects 273 of the 278 novel PE/COFF malware samples, leaving only 5 undetected. This small count (5) causes the standard error to be large because its reciprocal is used. In summary, SPEAD's detection accuracy for novel PE/COFF malware is significantly higher than the commodity anti-virus products in this experiment. This characteristic is very conducive to SPEAD's mission of detecting novel malware in email spear phishing attacks.

4.2.2.2 Adobe Reader Malware Detection Results

Table 10 summarizes the Adobe Reader malware detection metrics in the context of known and novel malware. Superficial analysis of SPEAD's novel malware detection rate indicates it is comparable to the commodity A/V products' rates, with less than two percentage points separating the top four performers.

Table 10: Adobe Reader Detection Results for Novel and Known Malware

System	Adobe Reader		
	Novel Detection Accuracy (%)	*Known* Detection Accuracy (%)	False Positive Rate (%)
SPEAD	88.79	51.74	0.96
Forefront	90.11	84.82	0.00
G Data	89.01	64.29	0.00
BitDefender	89.01	81.25	0.00
McAfee	41.76	43.75	0.00
AVG	72.16	59.45	0.00

To confirm this cursory analysis, a more in-depth analysis of the differences in means for the novel malware detection accuracies is shown in Figure 14. Tukey's HSD Method to test the differences in means is used to illustrate this point at a 95% confidence level. The pairwise comparisons indicate that the differences in means for SPEAD – Forefront, SPEAD – G Data, and SPEAD – BitDefender are not significantly different from zero because their confidence intervals include zero, and the p-values for each comparison are greater than 0.999. This means SPEAD's, Forefront's, G Data's, and BitDefender's novel Adobe Reader malware detection accuracies are very similar at a statistically significant level. This is confirmation that SPEAD's novel Adobe Reader malware detection rate is comparable to the best of the commodity A/V products in this experiment.

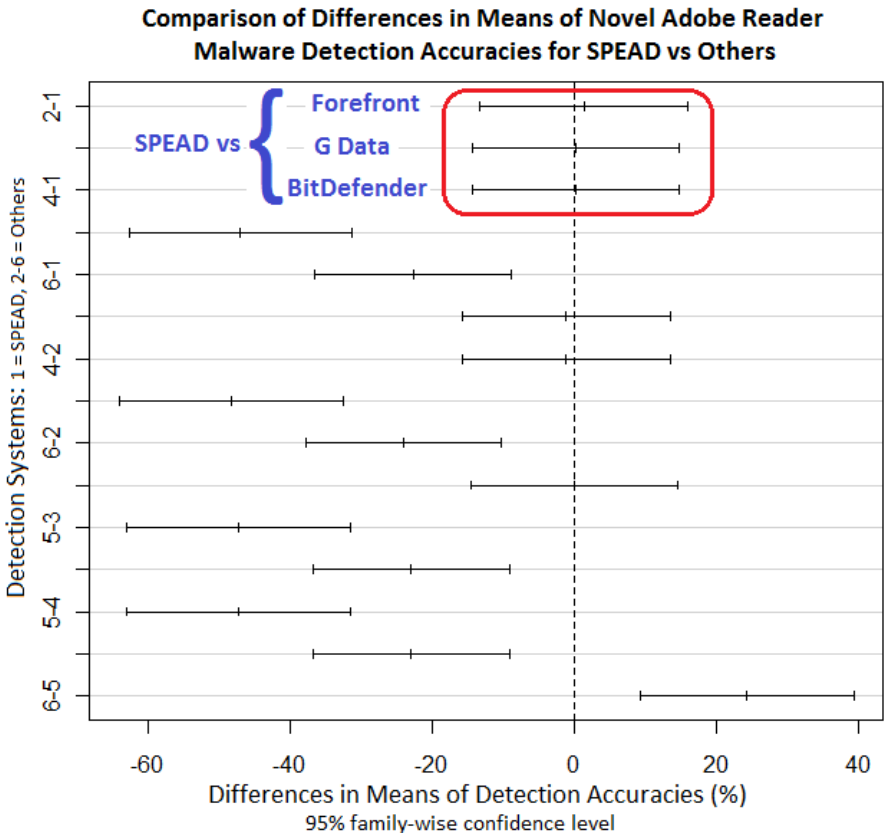
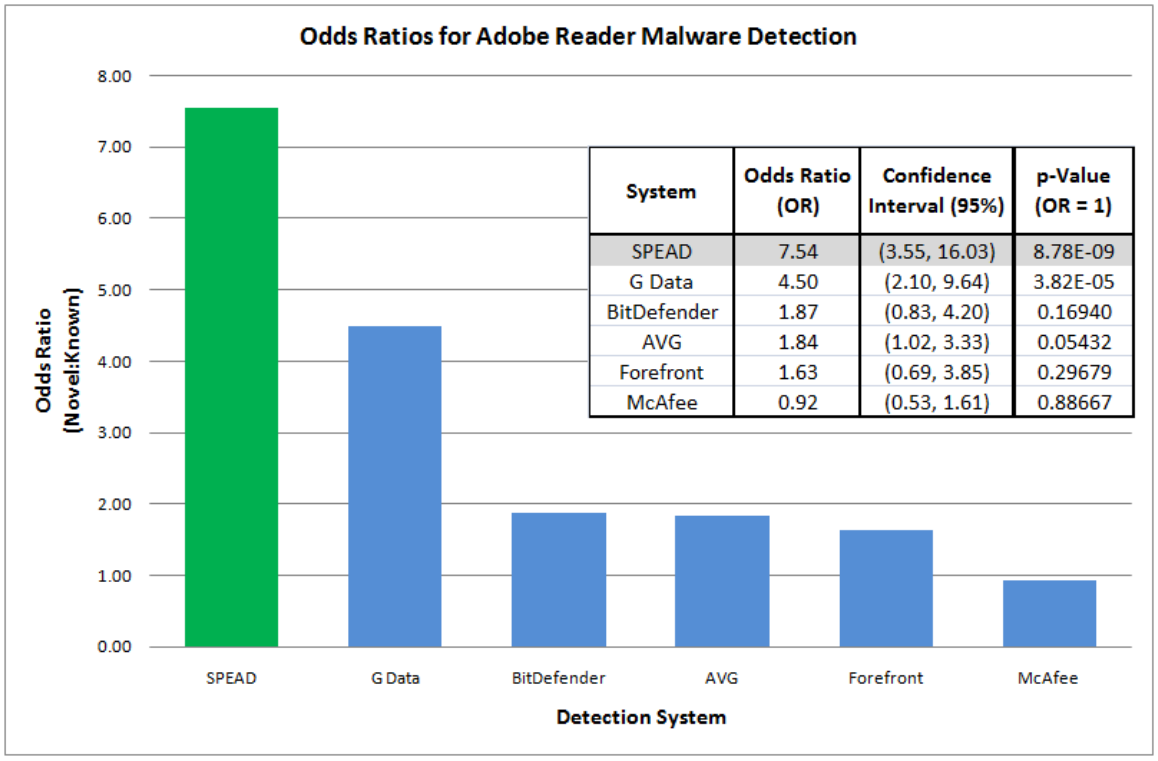


Figure 14: Comparison of Differences in Means of Adobe Reader Novel Malware Detection Accuracies for SPEAD versus Others

Furthermore, this data is viewed from one more angle. A close examination of Table 10 reveals a large discrepancy between SPEAD's novel and its known malware detection accuracies in comparison to the A/V systems' discrepancies. This fact has profound effects on the significance of ESCAPE's role in helping SPEAD detect novel Adobe Reader malware. The analysis is framed in this context: when a detection system detects Adobe Reader malware, what are the odds that the malware is novel?

The answer to this question is determined in the same way the PE/COFF data is analyzed to determine the odds ratio between two detection systems. Figure 15 illustrates the odds ratio for each detection system, and they are reported with 95% confidence intervals. The odds ratios are in terms of the odds of detection being of novel malware.

In other words, when SPEAD detects Adobe Reader malware, the odds of the malware being novel in nature is 7.54 times as large as the odds of it being known malware with a 95% confidence interval of 3.55 to 16.03 and a FET two-sided p-value of 8.78e-09. This p-value is the FET probability for the odds ratio being equal to 1, which is a test to see if the two populations are statistically the same. SPEAD's odds ratio and associated p-value provide conclusive evidence that its malware detection accuracy is statistically different between the novel and known malware populations. Even though SPEAD's odds ratio is 1.68 times higher than the next best, G Data, their confidence intervals do overlap. This fact is noteworthy, and it reduces but does not eliminate the significance of SPEAD's higher odds in detecting novel malware over known malware.



System	Odds Ratio (OR)	Confidence Interval (95%)	p-Value (OR = 1)
SPEAD	7.54	(3.55, 16.03)	8.78E-09
G Data	4.50	(2.10, 9.64)	3.82E-05
BitDefender	1.87	(0.83, 4.20)	0.16940
AVG	1.84	(1.02, 3.33)	0.05432
Forefront	1.63	(0.69, 3.85)	0.29679
McAfee	0.92	(0.53, 1.61)	0.88667

Figure 15: Odds Ratios of Novel:Known Adobe Reader Malware Detection

Therefore, this analysis leads to the conclusion that SPEAD is comparable to the best-performing commodity A/V products in terms of novel Adobe Reader malware detection outright, but when SPEAD detects malware, the odds of it being novel malware is higher. This is a highly desirable attribute in a spear phishing attack detection platform.

One reason for ESCAPE's success in detecting the novel nature of Adobe Reader malware in this experiment could be dependent upon the nature of the malware populations used in the malicious file corpus. The Adobe Reader malware labeled as *novel* is the malware collected from the two large, anonymous organizations. It is possible that the Adobe Reader malware used in attacks against these organizations use memory corruption exploitation techniques more often than the population of Adobe Reader malware outside these two organizations. It is also possible that Adobe Reader malware used in attacks against large organizations, in general, use memory corruption exploits more often.

Another reason for ESCAPE's successful focus on novel malware is because it is unencumbered by the need to detect all known malware. This is unlike the A/V products, whose commercial viability primarily hinges on its ability to detect as much malware as possible.

In terms of false positives, a reason for ESCAPE's false positives is due to its lack of appropriate exceptions. ESCAPE uses exceptions to handle legitimate application functionality that is not knowable when ESCAPE creates its database of signed code. It is possible the false positive Reader files attempt to use Reader functionality for which an exception has not been created for ESCAPE. Thus, ESCAPE detects this functionality as

anomalous and therefore malicious. This may also be true with the operating system and its need to run legitimate code that is unknown to ESCAPE. If this occurs while a file is being analyzed, ESCAPE will record this anomalous code execution attempt in its log file. The Python script may incorrectly attribute the anomaly to the file being analyzed, thus causing a false positive. Because SPEAD's false positive rate is still significantly low, it may not detract from SPEAD's ability to detect novel Reader malware at a high rate. An experiment with additional Adobe Reader files or on a real-world network may help characterize the significance, if any, of false positives for this file type.

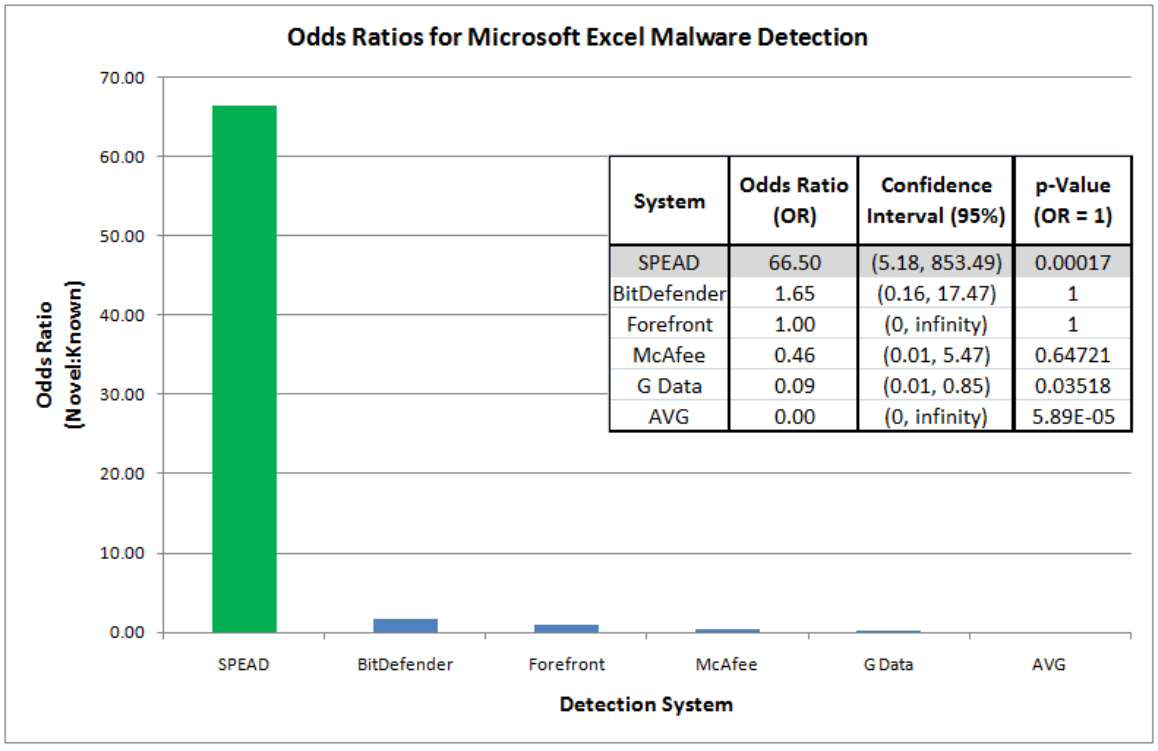
4.2.2.3 Microsoft Excel Malware Detection Results

Table 11 summarizes the Microsoft Excel malware detection metrics in the context of novel and known malware. SPEAD's novel malware detection accuracy appears to be significantly higher than all but Forefront's. The Excel sample size is very small compared to the PE/COFF and Adobe Reader malware sample sizes. With only 29 samples, Forefront's metrics appear anomalous, but they are impressive nonetheless. A more sophisticated analysis of the odds ratios is used again to draw conclusions about the likelihood of Excel malware detection being of novel malware.

Table 11: Microsoft Excel Detection Results for Novel and Known Malware

System	Microsoft Excel		
	Novel Detection Accuracy (%)	*Known* Detection Accuracy (%)	False Positive Rate (%)
SPEAD	90.48	8.33	0.67
Forefront	100.00	100.00	1.00
G Data	38.10	87.50	0.00
BitDefender	19.05	12.50	0.00
McAfee	76.19	87.50	0.00
AVG	0.00	75.00	0.00

The odds ratios for each detection system, in terms of the odds of detection being of novel Excel malware, are shown in Figure 16. Note that Forefront's odds ratio is only 1:1 even though it detects 100% of known and novel malware. This is not an anomalous result indicative of a failure in the odds ratio technique. In fact, the odds ratio for Forefront makes the point that when even though Forefront may detect all novel Excel malware, an analyst who needs to know if the malware is novel will do just as well flipping a coin to make that determination. The fact that Forefront's detection is exceptional is still a significantly valuable characteristic of a spear phishing detection platform, but so is a platform's ability to tell an analyst the probability that the malware is novel.



System	Odds Ratio (OR)	Confidence Interval (95%)	p-Value (OR = 1)
SPEAD	66.50	(5.18, 853.49)	0.00017
BitDefender	1.65	(0.16, 17.47)	1
Forefront	1.00	(0, infinity)	1
McAfee	0.46	(0.01, 5.47)	0.64721
G Data	0.09	(0.01, 0.85)	0.03518
AVG	0.00	(0, infinity)	5.89E-05

Figure 16: Odds Ratios of Novel:Known Microsoft Excel Malware Detection

When SPEAD detects Excel malware, the odds of the malware being novel in nature are 66.5 times as large as the odds of it being known malware with a 95% confidence interval of 5.18 to 853.49 and a FET two-sided p-value of 0.00017. Again, this p-value is the FET probability for the odds ratio being equal to 1, which is a test to see if the two populations are statistically the same. SPEAD's odds ratio and associated p-value provide conclusive evidence that its malware detection accuracy is statistically different between the novel and known malware populations. The wide confidence interval is again indicative of the small cell counts in the Excel detection 2x2 table, where 1 out of 8 known malware and 19 out of 21 novel malware are detected. This leaves two of the cells with counts of 1 and 2 for known malware detected and novel malware undetected, respectively. This causes a large standard error, and it can be mitigated with a larger sample size and, presumably, higher cell counts for the 2x2 table.

This analysis leads to the conclusion that SPEAD outperforms the other commodity A/V products, except Forefront, in terms of novel Microsoft Excel malware detection. Furthermore, the odds of malware detection being attributed to novel malware are extremely high only when SPEAD detects it. The strength of this conclusion is difficult to determine, however, because of the small sample size of Excel malware. Additional samples of known and novel malware are needed to be able to infer SPEAD's detection accuracy to a wider population.

Concerning false positives, the same reasoning applies here as it does in Section 4.2.2.3. ESCAPE may not be tuned appropriately to know all of Microsoft Excel's functionality.

4.2.2.4 Microsoft Word Malware Detection Results

Table 12 summarizes the Microsoft Word malware detection metrics in the context of novel and known malware. SPEAD's novel malware detection accuracy is clearly less than comparable to the commodity A/V detection rates. With only 22 samples, conclusions that can be drawn from this analysis are not strong. Still, the odds ratios are used to draw conclusions about the tendencies of Word malware detection being of novel malware.

Table 12: Microsoft Word Detection Results for Novel and Known Malware

System	Microsoft Word		
	Novel Detection Accuracy (%)	*Known* Detection Accuracy (%)	False Positive Rate (%)
SPEAD	22.22	7.69	0.00
Forefront	88.89	100.00	0.00
G Data	66.67	92.31	0.00
BitDefender	66.67	23.08	0.00
McAfee	22.22	100.00	0.00
AVG	11.11	69.23	0.00

The odds ratios for each detection system, in terms of the odds of detection being of novel Word malware, are shown in Figure 17. When SPEAD detects Word malware, the odds of the malware being novel in nature are 3.43 times as large as the odds of it being known malware with a 95% confidence interval of 0.26 to 45.03 and a FET two-sided p-value of 0.54416. This p-value suggests that any discrepancy between the observed odds ratio (3.43) and an odds ratio of 1 is likely due to chance. This is also true for BitDefender, which has the next highest odds ratio (2.67) and a p-value of 0.37616.

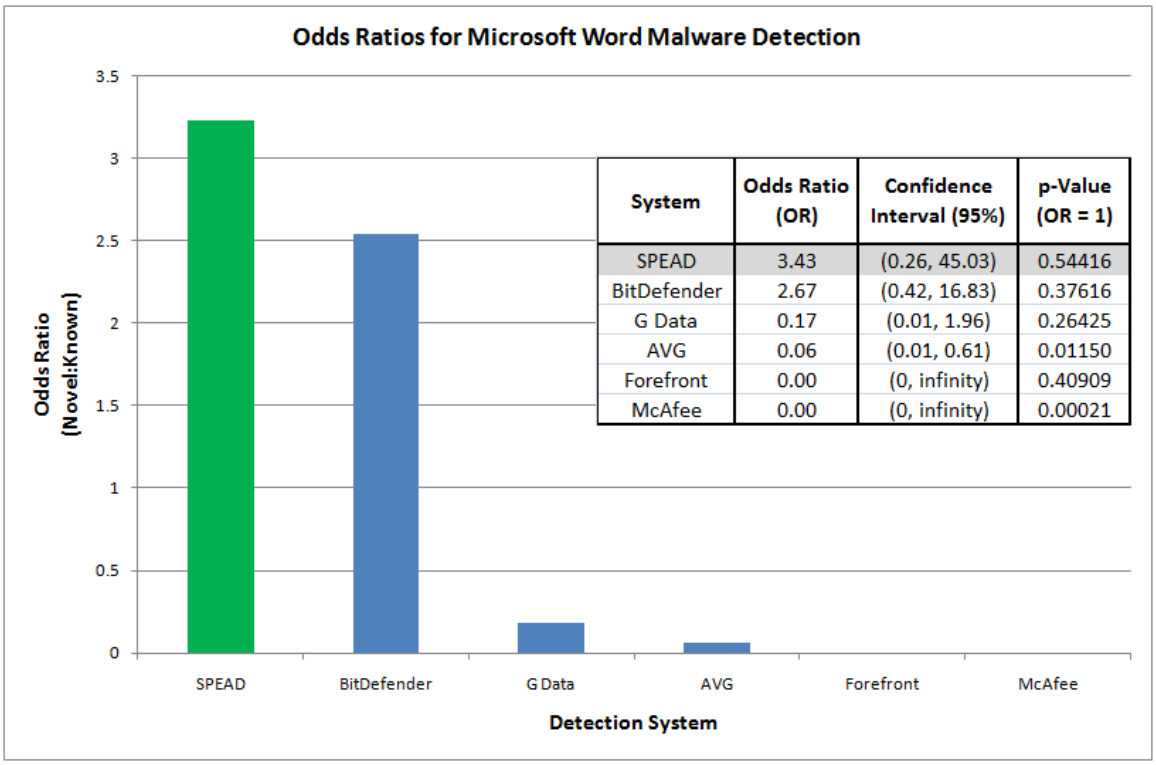


Figure 17: Odds Ratios of Novel:Known Microsoft Word Malware Detection

Even though SPEAD shows a tendency to focus its detection on novel Word malware, its significance is diminished in light of SPEAD's inaccuracy in detecting malicious Word files in this sample set (only 3 out of 22 detected overall) and the high probability of this favorable odds ratio being due to chance. Additional samples of known and novel malware are needed to characterize SPEAD's detection capabilities more accurately.

4.2.2.5 Microsoft PowerPoint Malware Detection Results

Table 13 summarizes the Microsoft PowerPoint malware detection metrics in the context of novel and known malware. SPEAD's lack of detecting any malicious samples in this small sample leads to an inconclusive analysis. With only 11 samples, conclusions that can be drawn from this analysis are weak. However, the detection rates by the

commodity A/V systems are moderate enough to suggest that these are, in fact, samples of malicious PowerPoint files. One reason for SPEAD's lack of detection could be due to PowerPoint-based attacks not relying on memory corruption exploits. If this is generally true, ESCAPE is not an ideal malware detection engine for PowerPoint malware. However, additional samples of known and novel PowerPoint malware are needed to characterize SPEAD's detection capabilities more accurately.

Table 13: Microsoft PowerPoint Detection Results for Novel and Known Malware

System	Microsoft PowerPoint		
	Novel Detection Accuracy (%)	*Known* Detection Accuracy (%)	False Positive Rate (%)
SPEAD	0.00	0.00	0.00
Forefront	50.00	100.00	0.00
G Data	75.00	85.71	0.00
BitDefender	75.00	85.71	0.00
McAfee	50.00	57.14	0.00
AVG	25.00	71.43	0.00

4.2.3 Comparing URL Detection Metrics

The detection of malicious URLs is a difficult problem to solve due to the fleeting nature of web-based exploits and how long they are viable and accessible before being taken down. Table 14 is a good illustration of the fact that the approach to detecting malicious URLs varies widely. Even though G Data achieves the highest detection accuracy (29.69%), it also demonstrates the highest false positive rate (25.97%) by far. This suggests G Data's detection algorithm or content filtering is, perhaps, overly generic in its detection. While this is good for malicious detection rates, it also causes G Data to label numerous URLs as malicious even though they are not. McAfee clearly displays

the best balance of detection accuracy (25.26%, second highest) with a relatively low false positive rate (1.06%).

Table 14: URL Detection Results for All Platforms

System	URLs	
	Detection Accuracy (%)	False Positive Rate (%)
SPEAD	2.92	0.71
Forefront	0.94	0.00
G Data	29.69	25.97
BitDefender	0.00	0.00
McAfee	25.16	1.06
AVG	0.24	0.00

There are two important discussion items that are not readily apparent by looking at only the detection metrics:

1) Malware or malicious code must be actively hosted from a web site for ESCAPE to detect it. This is a fundamental design feature of ESCAPE, but it makes testing and comparison difficult to do with certainty. Theoretically, the longer a URL is known to be malicious, the more likely A/V products and content filtering engines are to detect it. Conversely, the longer a URL is known to be malicious, the more likely it is to terminate or limit its active malware hosting due to the ever-increasing attention the web site receives from those who are not its intended victims. This means ESCAPE will not detect the malicious web site where a commodity A/V product might through the use of a known bad list. Many of the malicious URLs used in this experiment are more than a month old, which is not an ideal testing scenario where the focus is on novel malicious code exploits.

2) Out of the 56 URLs detected in each of the three tests, 35 of them are due to the download of a PE/COFF file that is deemed to be malicious by MaTR. These malicious PE/COFF files are saved on the file share. This fact alone provides a malware or network intrusion analyst a tangible and significant value that commodity A/V does not. Additionally, the fact that ESCAPE detects a URL as malicious is a strong indicator that the web site is actively exploiting victims at the time the email is received that contained the URL. This immediate feedback, proven by dynamic analysis, is valuable information for an analyst. Therefore, SPEAD provides two capabilities that the other A/V products do not: malicious PE/COFF file downloads and immediate confirmation of active and current malicious websites.

The reasons for false positives in this experiment are the same reasons already discussed in Sections 4.2.2.2 and 4.2.2.3. There may be Internet Explorer functionality of which ESCAPE is not aware. Full feature testing may reduce the false positive rate, and it is a consideration for future research.

4.3 Results and Analysis of Experiment 4

The results of this experiment are used to characterize SPEAD's latency, which is the time it takes SPEAD to receive an email and to make a determination whether the email is malicious or not. It is important to note that SPEAD's malware detection rates are not being evaluated in this experiment. This is because MaTR's and ESCAPE's execution environments already process at their maximum configured speeds when a

queue exists for files or URLs waiting to be analyzed. Thus, a larger queue does not affect their detection capabilities.

This experiment uses three sizes of file and URL sample sets based on the email throughput being tested. The appropriate number of files and URLs are selected to allow for a one-minute duration of emails being sent:

- 12 files and 240 URLs are selected for the low throughput of 12 attachments/minute and 240 URLs /min
- 60 files and 1,000 URLs are selected for the expected throughput of 60 attachments/minute and 1,000 URLs /min
- 300 files and 1,500 URLs are selected for the max throughput of 232 attachments/minute and 1,422 URLs /min

These samples are randomly selected from the malicious and non-malicious PE/COFF files, non-PE/COFF files, and URLs. Each file type is as equally represented as possible based on the sample size. The file latency results are analyzed first, followed by the URL latency results.

4.3.1 File Latency Results and Analysis

The test for each throughput speed is repeated twice for a total of three tests at each throughput speed. The average latency results for each throughput are calculated, and the file results are displayed in Figure 18. These plots show a linear growth trend after an initial slow-growth period, especially when the sample size is large as it is in the maximum throughput test. These initial slow-growth trends are due to the rapid static analysis responses offered by MaTR for the PE/COFF files.

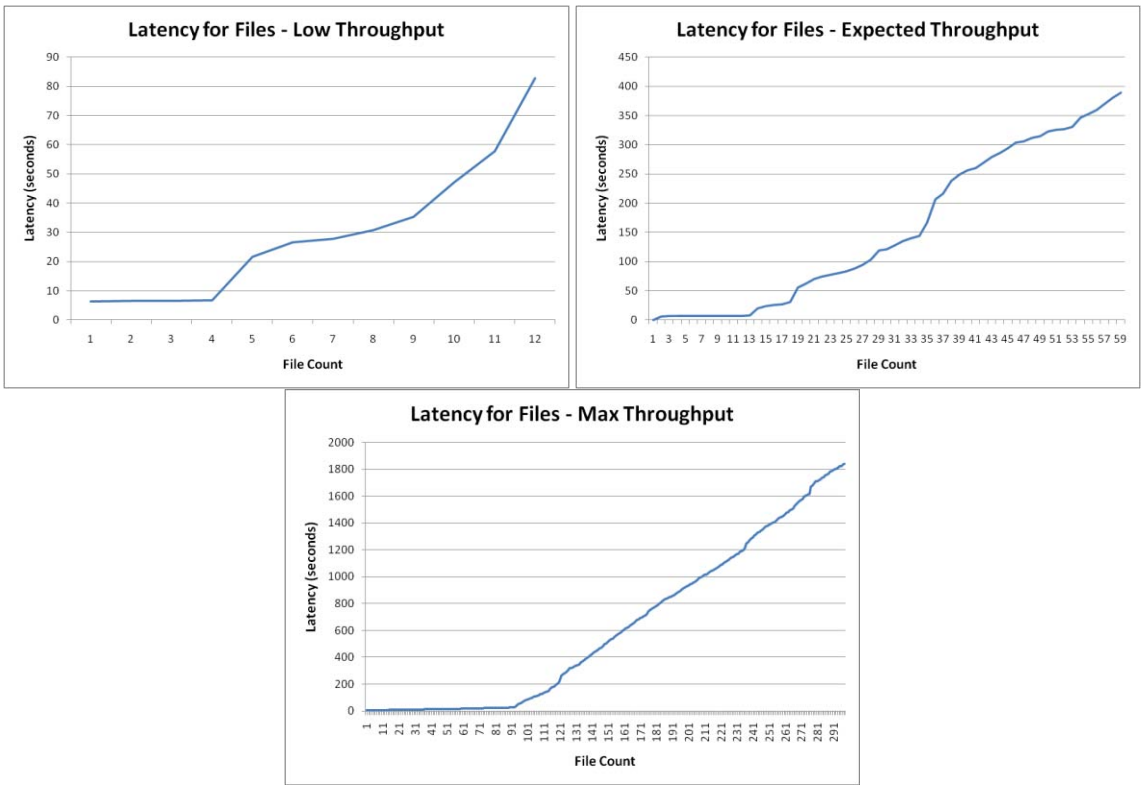


Figure 18: Latency Results for All Files at Each Email Throughput Speed

In order to quantify SPEAD's file processing latency more accurately, the file results are segregated into PE/COFF and non-PE/COFF latency responses. Figure 19 shows the plots of the non-PE/COFF latency results. A linear trend line and associated R^2 statistic are calculated and displayed on each plot. The R^2 statistic represents the percentage of the total system response variation that is explained by the explanatory variable [RaS02], which, in the case of a simple linear regression, is the slope of the line. The high R^2 statistic supports the linear trend line for the expected (98.03%) and maximum (99.56%) email throughput plots. Because the linear trend line (in red) for the low throughput plot appears not to fit well (R^2 = 83.97%), an exponential model (in green) is offered as a better fit (R^2 = 94.19%) for the small sample size. Based on these plots, the following qualitative observations are made:

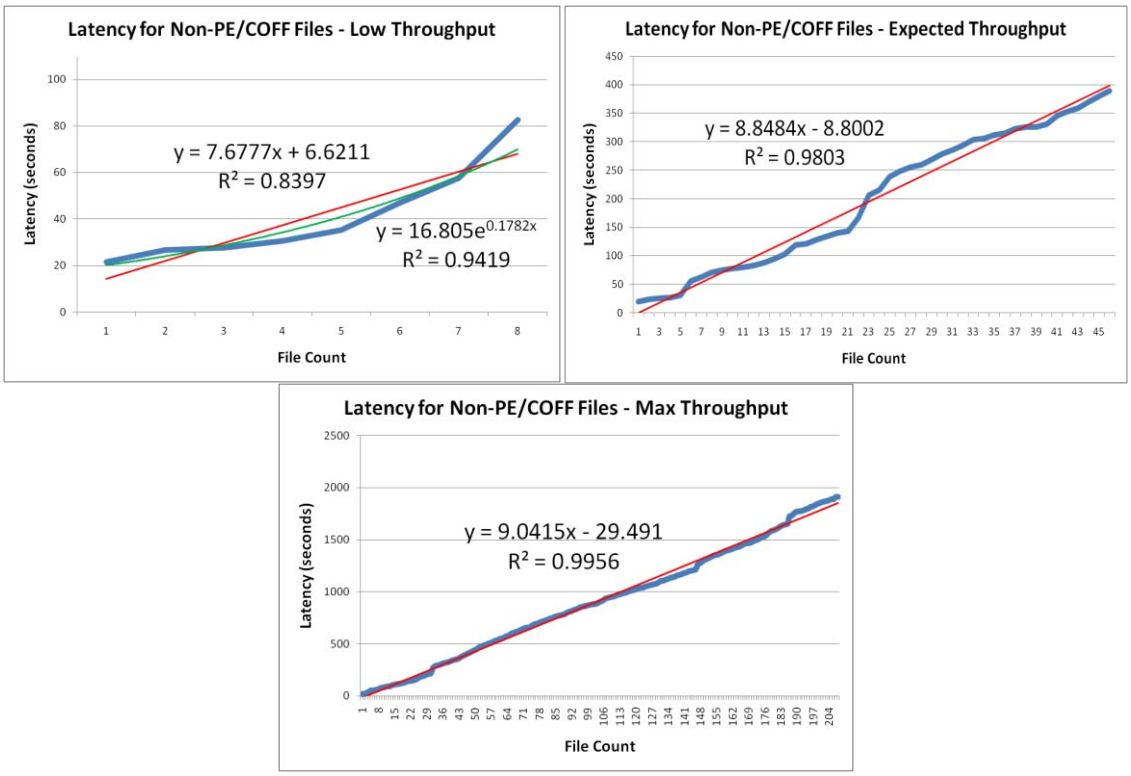


Figure 19: Latency Results for Non-PE/COFF Files at Each Email Throughput Speed

- A perfectly horizontal line means new files received by SPEAD would always have the same expected wait time regardless of the current queue of files to be processed. Because these lines show a linear growth, this means there is an increase in latency for each new file received by SPEAD. This demonstrates, essentially, a queuing delay within SPEAD. Analogous to a queuing delay in network routers when packets arrive faster than the router can process them, SPEAD experiences a queuing delay when emails arrive at a rate faster than they are processed. This queue consists of the unanalyzed files and URLs in the malware detection database.
- The slopes of the trend lines translate to the expected increase in wait time, and these increases in latency are estimated to be 7.68 seconds, 8.85 seconds, and 9.04

seconds for each new file introduced in a low, expected, and maximum throughput environment, respectively. The exponential model can also be used as an estimator of the expected file latency at low email throughputs, but its equation is not as intuitive as the linear equation for the purpose of cursory estimation of latencies. Additionally, because the expected and maximum throughput models are distinctly linear with larger samples sizes, it is reasonable to assume the low throughput model is truly linear. The small sample size (8 files) may not be sufficient to produce a strong linear trend over time.

The plots of the PE/COFF file latency results are shown in Figure 20. The low and expected throughput's linear trend lines both show a predominately horizontal tilt, indicating a nearly ideal growth in wait time, or lack thereof. The maximum throughput plot shows both an exponential and linear growth trend for comparison. The following qualitative observations are made:

- The R^2 statistic (78.5%) for the low throughput is counterintuitive. Judging by the difficulty in differentiating the linear trend line from the data's plotted line, it appears as though the R^2 statistic should be closer to 100%. The R^2 statistic is somewhat lower in this case because there is very little total variation in latency across these four files. The linear equation explains 78.5% of this small total variation, which results in a very good-fitting model without a R^2 of 100%.
- The expected increase in latency for each new PE/COFF file received by SPEAD is 0.07 seconds for the low and expected workloads.

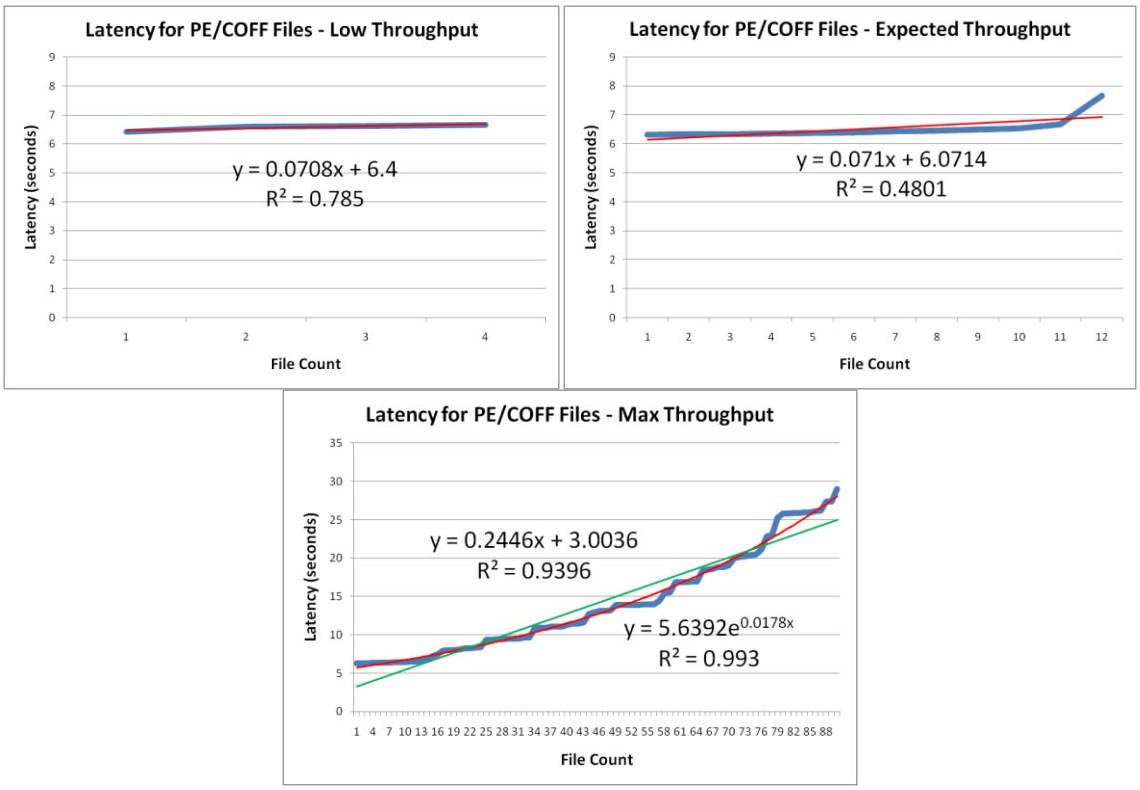


Figure 20: Latency Results for PE/COFF Files at Each Email Throughput Speed

- By visual inspection of the maximum throughput plot, the exponential trend line appears to be the better fit. The expected increase in latency for each new PE/COFF file received by SPEAD is $5.64e^{0.0178}$ seconds, or about 0.25 seconds if the growth is linear, for the maximum throughput. There appears to be a saturation point somewhere between the expected and maximum throughput speeds where the latency growth begins to increase above the 0.07/file linear rate.
- If the latency growth is truly linear for the maximum throughput environment, then there are several possible reasons to explain why the plot appears to be exponential:
    a) The VBscript responsible for calculating the latency metric polls the database every five seconds. With enough samples over time, the average

relative growth in latency between samples (i.e., the slope of the line) covers up the effect of this polling interval on the observed latency. However, at the start of an experiment, no latency less than five seconds is possible because the script will not poll for responses until the five-second mark. Because of this, the first 14 samples at the start of the maximum throughput test all have latencies in the six-second range, giving the plot the appearance of exponential growth as samples 15 and beyond resume a linear growth in latency.

b) There is a mix of malicious and non-malicious PE/COFF files in the sample set for the maximum throughput tests. MaTR's analysis time may be sensitive to files that are malicious due to the extra traversing of the decision tree needed to classify the malware's type as backdoor, Trojan, worm, etc. If this is the case, then a run of consecutive malicious PE/COFF files may slightly skew the latency growth trend line. Coincidentally, the last 19 files analyzed in the maximum throughput scenario are all classified as malicious, which may have an additive effect on the latency growth trend and give it the appearance of an exponential turn upwards on the graph towards the latter samples.

c) The underlying ESX server may be reaching a processing speed limit due to the immediate influx of processing needed for 17 virtual images (1 MaTR and 16 ESCAPE) to analyze files and URLs at the maximum throughput rate. This may cause noticeable increases in latency for

relatively small growth trends like that shown by the rapid PE/COFF file processing.

4.3.2 URL Latency Results and Analysis

The URL latency plots are averages of three tests, just as the file latency plots are. These plots are displayed in Figure 21. Linear trend lines and R^2 statistics are added to these plots as well as the linear equation used to formulate the trend line. These linear trend lines fit the plots very well in all three test scenarios with R^2 statistics over 99% in each plot. Thus, a direct comparison of the slopes of the equations can be made. The expected increases in latency for a new URL received by SPEAD are 3.78 seconds, 3.90 seconds, and 4.04 seconds, respectively, for the low, expected, and maximum throughput conditions.

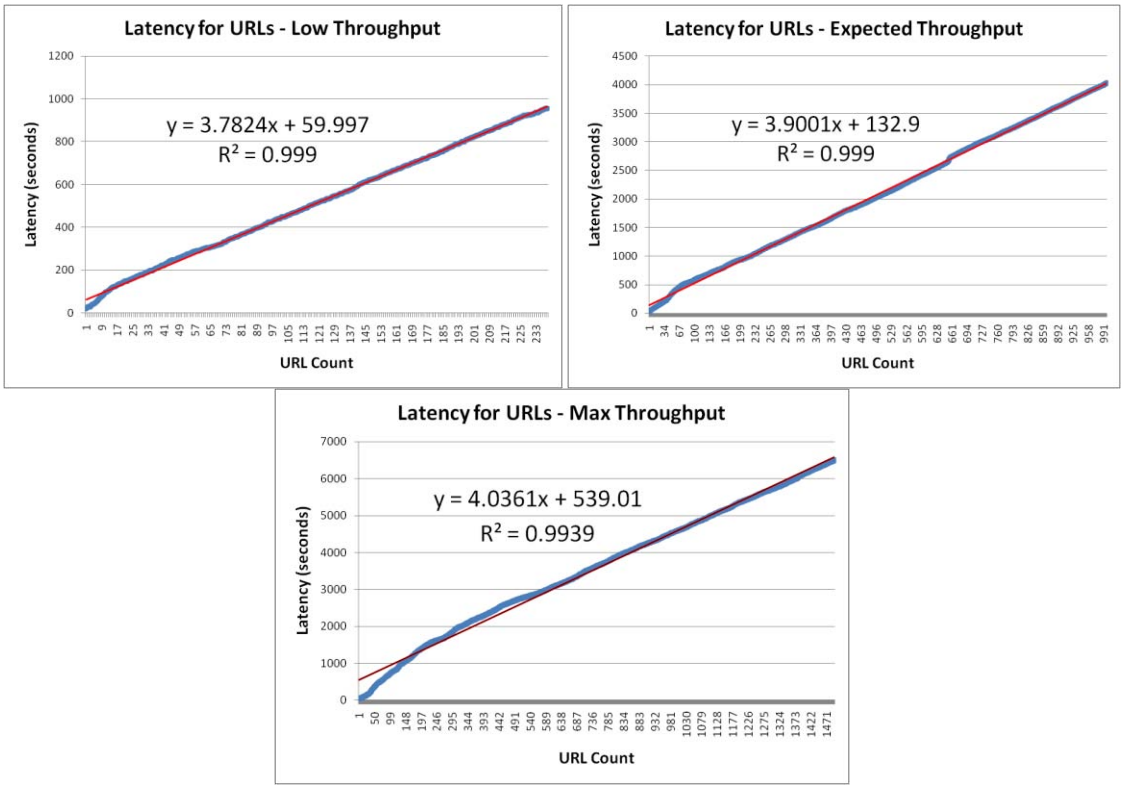


Figure 21: Latency Results for URLs at Each Email Throughput Speed

It is noteworthy that there is only a 0.12 second latency increase going from the low to the expected throughput, but there is a 0.14 second latency increase going from the expected to the maximum throughput. This is despite the growth in URLs with the former (760 URL increase) being much greater than the growth in URLs with the latter (500 URL increase). This indicates a saturation point for URL processing at throughputs somewhere between 1,000 and 1,500 URLs per minute.

4.3.3 Overall Latency Analysis

SPEAD's non-PE/COFF file processing does no better than a latency increase of about 7.7 seconds per additional file received even at the low email throughput. Compared to the best URL processing latency of about a 3.8 second increase per additional URL, it is noteworthy that the non-PE/COFF file processing latency grows at approximately twice the rate. This may be due to the file-based applications needing more processing bandwidth or more memory than Internet Explorer, thus requiring a little more time to open each file and also to terminate the application upon analysis completion.

SPEAD's PE/COFF file processing is significantly faster than its non-PE/COFF file and URL processing. The evidence strongly supports the notion that static analysis is much quicker than dynamic analysis. SPEAD appears to be able to handle much higher email throughput rates of PE/COFF files.

Finally, the data is viewed in the context of the research hypothesis that SPEAD can operate under any sustained traffic workload and still detect novel malware in one hour or less. Table 15 is the summary of this data. These calculations assume the email

throughput workload is constant, but traffic that is bursty would have some idle time to allow SPEAD an opportunity to continue processing queued files and URLs without introducing new items. For PE/COFF inputs, SPEAD can handle a sustained maximum workload for over an hour before the latency reaches one hour per item. For non-PE/COFF files, even a low, sustained throughput workload pushes SPEAD's latency per file to the one hour mark before 40 minutes. The URL input is slightly better, lasting until almost 80 minutes at a low workload before SPEAD's latency reaches one hour per URL. This analysis supports a conclusion that the research hypothesis concerning SPEAD's latency is not correct. It is important to note that the point of this experiment is to quantify SPEAD's non-optimal runtime performance configuration for future comparison in case SPEAD's performance is enhanced for optimal runtime speeds.

Table 15: Expected Time to Reach 1 Hour Latency

Input Type	Low Throughput			Expected Throughput			Max Throughput		
	Latency Growth Rate (seconds)	# of Items until 1 Hour Latency	Time to Reach 1 Hour Latency (minutes)	Latency Growth Rate (seconds)	# of Items until 1 Hour Latency	Time to Reach 1 Hour Latency (minutes)	Latency Growth Rate (seconds)	# of Items until 1 Hour Latency	Time to Reach 1 Hour Latency (minutes)
PE/COFF	0.07	50704.23	4225.35	0.07	50704.23	845.07	0.25	14693.88	63.34
Non-PE/COFF	7.68	468.75	39.06	8.85	406.78	6.78	9.04	398.23	1.72
URL	3.78	952.38	79.37	3.90	923.08	15.38	4.04	891.09	3.84

4.4 Summary

This chapter presents and analyzes the data collected from the four experiments undertaken by this research. The results of Experiment 1 are discussed, and the analysis conclusions are used to configure ESCAPE for optimal malware detection accuracy with a consideration for speed. The results of Experiments 2 and 3 are analyzed, and many conclusions are made. Namely, the original code for SPEAD is validated, and SPEAD

proves to be a viable option as a complementary email-based malware detection framework that focuses primarily on novel malware, especially PE/COFF, Adobe Reader, and Microsoft Excel malware. Furthermore, SPEAD's URL detection metrics and unique detection capabilities are discussed and quantified. Finally, SPEAD's latency results are presented and analyzed in order to characterize SPEAD's performance in terms of the research hypothesis.

V. Conclusions

This chapter summarizes the overall conclusions drawn from this research. Section 5.1 compares the four research goals with the experimental results to determine if the research goals and hypotheses are met. The significance of this research is outlined in Section 5.2. Finally, suggestions for future work to extend this research are provided in Section 5.3.

5.1 Research Conclusions

5.1.1 Goals #1 and #2: Construct a spear phishing detection system

The first goal of this research is to construct an email collection and processing system to obtain emails, parse them for files and Uniform Resource Locators (URLs), and insert appropriate information into a database for automated malware analysis. The second research goal is to modify the execution environment of two malware detection algorithms, ESCAPE and MaTR (Malware Type Recognition), to interact with this database for file/URL download and the upload of detection results. Original code is written to create a framework that accomplishes these first two goals. This framework is called the SPEar phishing Attack Detection system (SPEAD). Experiment 1 is used to optimize ESCAPE's analysis wait time to 10 seconds. The results of Experiment 2 are used to verify that all files and all URLs are successfully processed and inserted into the database. This validates the correctness and effectiveness of the original code. Thus, the first two research goals are achieved.

5.1.2 Goal #3: Compare this system to the current industry standard

The third goal of this research is to collect malware detection metrics for SPEAD and commodity anti-virus products to determine SPEAD's effectiveness in detecting novel email-borne malware. Experiments 2 and 3 provide ample metrics and data to perform a comparison. In comparison to the commercial products, SPEAD is the best performer for the overall detection of all malicious Portable Executable and Common Object File Format (PE/COFF) files (99.68% true positive rate, 0.39% false positive rate) as well as novel PE/COFF malware (98.2% true positive rate).

SPEAD's performance is also statistically comparable to the anti-virus products in terms of the detection of novel Adobe Reader malware with a 88.79% true positive rate and the fact that the pairwise differences in means between SPEAD and the other three top performers is not significantly different from zero (two-sided p-values greater than 0.999 for each pairwise comparison).

Furthermore, SPEAD demonstrates unique advantages in terms of its statistically strong tendency to focus its detection on novel malware only. Specifically, the odds of a SPEAD malware detection being attributed to novel malware are as follows:

- For PE/COFF files, 42.1:1 odds in favor of SPEAD detecting novel PE/COFF malware over the next best-performing anti-virus product (95% confidence interval is 16.84 to 105.14 with two-sided p-value less than 2.2e-16).
- For Adobe Reader files, 7.54:1 odds that when SPEAD detects Reader malware, the malware is novel (95% confidence interval is 3.55 to 16.03 with two-sided p-value of 8.78e-9). The next best performer is G Data with 4.5:1 odds (95% confidence interval of 2.10 to 9.64 with two-sided p-value of 3.82e-5).

- For Microsoft Excel files, 66.5:1 odds that when SPEAD detects Excel malware, the malware is novel (95% confidence interval is 5.88 to 853.49 with two-sided p-value of 0.00017). The next best performer is BitDefender with 1.65:1 odds (95% confidence interval of 0.16 to 17.47 with two-sided p-value of 1).

Additionally, SPEAD offers two unique benefits from analysis of email-borne URLs: 1) automated PE/COFF malware download from malicious URLs (35 downloads during Experiment 2, and 2) near real time confirmation of active malicious web sites (56 sites detected as malicious). The hypothesis that two malware detection algorithms can co-exist in an email context while outperforming commodity anti-virus products is mostly confirmed. Full confirmation is lacking due to SPEAD's lack of effectiveness in detecting novel Microsoft Word and PowerPoint malware at a sufficient rate with this limited sample set. Even though the hypothesis is not fully confirmed, the third research goal is still achieved, which aimed to quantify SPEAD's effectiveness in relation to commodity A/V solutions.

5.1.3 Goal #4: Characterize the detection latency of this system

The fourth goal of this research is to characterize SPEAD detection latency while using an approximated Air Force base's email traffic workload. It is important to note that SPEAD does not have an optimal runtime performance configuration. This research goal simply aims to take a snapshot of SPEAD's latency characteristics as a gauge for future work in this area, if necessary. Experiment 4 performs the necessary tests to quantify SPEAD's latencies in varying email traffic workloads. SPEAD's latencies in

the worst case scenario (i.e., email throughput of 300 files/minute and 1,500 URLs/minute) are as follows:

- For PE/COFF files, analysis is expected to cause one-hour latencies in about 63.34 minutes.
- For non-PE/COFF files, analysis is expected to cause one-hour latencies in about 1.72 minutes.
- For URLs, analysis is expected to cause one-hour latencies in about 3.84 minutes.

While the fourth research goal is achieved, the hypothesis that SPEAD can effectively detect malware under any sustained workload within one hour is not confirmed.

5.2 Significance of Research

This research provides the Air Force and other large organizations with the capability for fully automated detection of email spear phishing attacks indicative of cyber espionage. No other public framework or product exists that combines malware detection algorithms for the sole purpose of autonomously identifying previously unknown malicious software *and* malicious web site links delivered through emails.

SPEAD can be implemented in a plug-and-play manner for any network enterprise employing Microsoft Exchange as its email service or that can provide emails to a Microsoft Outlook email client. Its passive network presence and near real time detection provide network security analysts with the unique benefit of immediate cyber espionage situational awareness. Additionally, SPEAD is a good complement to

traditional anti-virus and anti-spam solutions because of its unique ability to identify the novel malware attacks many anti-virus solutions and anti-spam engines struggle to find in email attachments and URLs.

5.3 Recommendations for Future Research

A myriad of extensions to this research are viable. [MeM11] proposes an automated framework for the detection of cyber espionage events on a network, and SPEAD's spear phishing detection capabilities can be implemented within such a framework.

SPEAD can also be extended to include an inline network configuration, where SPEAD's capabilities are enhanced for automated *prevention* of novel email attacks. This requires an optimization study to determine the appropriate configuration and coding improvements to decrease SPEAD's latency under varying workloads. Included in this optimization study could be the addition of more ESCAPE and MaTR virtual clients to maximize parallel processing.

Since SPEAD is a framework for integrating malware detection algorithms, this research can be extended to include other cutting edge detection algorithms. Additional testing could include scenarios where SPEAD's malware detection algorithms are complemented by varying anti-virus products to determine the best complementary configuration for both known and novel malware detection.

An observational study using SPEAD on an operational network would undoubtedly test SPEAD's detection of truly novel and previously unknown email-based

attacks. This study would also validate SPEAD's URL parsing effectiveness with real-world emails and URLs.

Additionally, the following specific enhancements can be coded, tested, and evaluated:

- Check all incoming URLs against a known good whitelist and known bad blacklist to reduce the amount of unnecessary URL analysis
- Public Key Infrastructure validation: compare the sender of digitally signed and/or encrypted emails with what is represented in the signature or public key
- Parse emails for archive file formats such as .zip and .rar files; automatically decompress these files for analysis
- Parse files for embedded documents (i.e., .pdf embedded in a Word document)
- Create a software agent to click on, open, or download items within documents or on web sites for more in-depth dynamic analysis
- Configure SPEAD to revert ESCAPE clients to snapshots every time a malicious file/URL is detected
- Configure SPEAD to implement prioritization of analysis; evaluate the performance improvement of SPEAD malware detection algorithms re-scanning old file/URLs during idle time
- Create original exploits for each file type for testing SPEAD's novel malware detection; same can be applied to web-based attack vectors
- Capture ESCAPE's detection metrics with enough granularity to compare malware detections on Windows XP to those on Windows Vista and/or Windows

7 to determine the effectiveness of Windows native memory corruption protection mechanisms

- Perform full feature testing of ESCAPE-protected applications to reduce the false positive detection rate
- Evaluate all ESCAPE configuration factors to determine an optimal configuration for speed of detection without sacrificing accuracy
- Use obfuscated, packed, or compressed malware to evaluate detection limitations of SPEAD versus commodity anti-virus

Bibliography

[ACK+04] T. Abou-Assaleh, N. Cercone, V. Keselj and R. Sweidan. "Detection of New Malicious Code Using N-grams Signatures," *Proceeding of the 2nd Annual Conference on Privacy, Security, and Trust*. pp. 193-196, 2004.

[ACKS04] T. Abou-Assaleh, N. Cercone, V. Keselj and R. Sweidan. "N-gram-based detection of new malicious code," *Proceedings of the 28th Annual International Computer Software and Applications Conference - Workshops and Fast Abstracts (COMPSAC '04)*. Vol. 2, pp. 41-42, Washington, D.C.: IEEE Computer Society, 2004.

[AHH+02] F. Apap, A. Honig, S. Hershkop, E. Eskin and S. Stolfo. "Detecting malicious software by monitoring anomalous Windows registry accesses," *Recent Advances in Intrusion Detection: Lecture Notes in Computer Science*. Vol. 2516, pp. 36-53, Berlin: Springer, 2002.

[ArT00] W. Arnold and G. Tesauro. "Automatically generated Win32 heuristic virus detection," *Virus Bulletin Conference.* pp. 51-60, 2000.

[BaH02] M. Bhattacharyya and S. Hershkop. "MET: An experimental system for Malicious Email Tracking," *Proceedings of the 2002 Workshop on New Security Paradigms*. pp. 3-10, New York: ACM, 2002.

[BDG+10] A. Bergholz, J. De Beer, S. Glahn, M. F. Moens, G. Paaß and S. Strobel. "New filtering approaches for phishing email," *Journal of Computer Security*. Vol. 18, No. 1, pp. 7-35, 2010.

[Ber09] A. Bergholz. "AntiPhish: Lessons Learnt," *Proceedings of the ACM SIGKDD Workshop on CyberSecurity and Intelligence Informatics*. New York: ACM, 2009.

[BOA+07] M. Bailey, J. Oberheide, J. Andersen, Z. Mao, F. Jahanian and J. Nazario. "Automated classification and analysis of internet malware," *Proceedings of the 10th International Conference on Recent Advances in Intrusion Detection*. Berlin: Springer-Verlag, pp.178-197, 2007.

[ChJ04] M. Christodorescu and S. Jha. "Testing malware detectors," *Proceedings of the 2004 ACM SIGSOFT International Symposium on Software Testing and Analysis*. pp. 34-44, New York: ACM, 2004.

[CJS+05] M. Christodorescu, S. Jha, S. Seshia, D. Song and R. Bryant. "Semantics-aware Malware Detection," *Proceedings of the IEEE Symposium on Security and Privacy*. pp. 32-46, 2005.

[CKO+08] Y. Choi, I. Kim, J. Oh and J. Ryou. "PE file header analysis-based packed PE file detection technique (PHAD)," *Proceedings of the International Symposium on Computer Science and its Applications*. pp. 28-31, Washington, D.C.: IEEE Computer Society, 2008.

[COP10] J. Crain, L. Opyrchal and A. Prakash. "Fighting phishing with trusted email," *Proceedings of the 2010 International Conference on Availability, Reliability and Security*. pp. 462-467, 2010.

[Dat10] "Data Execution Prevention," Retrieved 6 June 2010 from http://support.microsoft.com/kb/875352.

[DGL09] J. Dai, R. Guha and J. Lee. "Feature Set in Data Mining Techniques for Unknown Virus Detection—Comparison Study," *Proceedings of the 5th Annual Workshop on Cyber Security and Information Intelligence Research: Cyber Security and Information Intelligence Challenges and Strategies*. No. 56, New York: ACM, 2009.

[Dix10] B. Dixon. Malware Analyst and Creator of blog.9bplus.com. Personal Correspondence. 17 December 2010.

[DJB+09] J. Ding, J. Jin, P. Bouvry, Y. Hu and H. Guan. "Behavior-based Proactive Detection of Unknown Malicious Codes," *4th International Conference on Internet Monitoring and Protection*. pp. 72-77, Washington, D.C.: IEEE Computer Society, 2009.

[DRP+10] T. Dube, R. Raines, G. Peterson, K. Bauer, M. Grimalia, and S. Rogers. "Malware Type Recognition and Cyber Situational Awareness," *2nd IEEE International Conference on Social Computing (SocialCom 2010)*. pp. 938-943, 2010.

[Eps08] K. Epstein. "U.S. Is Losing Global Cyber War, Commission Says," *BusinessWeek.* 7 December 2008. Retrieved on 23 January 2010 from:http://www.businessweek.com/bwdaily/dnflash/content/dec2008/db20081 27_817606.htm?chan=top+news_top+news+index+-+temp_ dialogue+ with+readers.

[FST07] I. Fette, N. Sadeh and A. Tomasic. "Learning to detect phishing emails," *Proceedings of the 16th International Conference on World Wide Web*. pp. 649-656, New York: ACM, 2007.

[GET08] B. Grow, K. Epstein and C. Tschang. "The New E-spionage Threat," *BusinessWeek.* 10 April 2008. Retrieved on 23 January 2010 from: http://www.businessweek.com/magazine/content/08_16/b4080032218430.htm.

[HeJ06] O. Henchiri and N. Japkowicz. “A feature selection and evaluation scheme for computer virus detection,” *Proceedings of the Sixth International Conference on Data Mining*. pp. 891-895, Washington, D.C.: IEEE Computer Society, 2006.

[Hin08] M. Hines. “Cyber-espionage Moves into B2B,” *InfoWorld.* 15 January 2008. Retrieved on 10 June 2010 from: http://www.infoworld.com/t/business/cyber-espionage-moves-b2b-546.

[HoB05] G. Hoglund and J. Butler. *Rootkits: Subverting the Windows Kernel*. Addison-Wesley Professional, 2005.

[Hub06] “Hub Transport Server Role: Overview,” 14 September 2006. Retrieved on 5 January 2010 from: http://technet.microsoft.com/en-us/library/bb123494(EXCHG.80).aspx.

[IdM07] N. Idika and A.P. Mathur. “A Survey of Malware Detection Techniques,” *Purdue University*, 2007.

[Jac10] K. Jackson. “Spear-phishing attacks out of China targeted source code, intellectual property,” 13 January 2010. Retrieved on 21 November 2010 from: http://www.darkreading.com/database-security/167901020/security/attacks-breaches/222300840/index.html

[KeA94] O. Kephart and W. C. Arnold. “Automatic extraction of computer virus signatures,” *Proceedings of the 4th Virus Bulletin International Conference*. pp. 178-184, 1994.

[Kei10] G. Keizer. “Google hackers behind Adobe Reader PDF zero-day bug, Symantec warns,” 15 September 2010. Retrieved on 21 November 2010 from: http://news.techworld.com/security/3239606/google-hackers-behind-adobe-reader-pdf-zero-day-bug-symantec-warns/

[Ken10] H. Kenyon. ”Air Force starts long trek to Windows 7,” 4 November 2010. Retrieved on 13 October 2010 from: http://defensesystems.com/articles/2010/11/04/air-force-begins-transition-to-windows-7.aspx.

[Kim10] W. Kimball, Faculty Research Assistant, Center for Cyberspace Research, Air Force Institute of Technology. “ESCAPE Vista and Win7: An Autonomic and Cryptographic Kernel Software Protection System.” Written Instruction Manual for the Use of ESCAPE. Air Force Institute of Technology, Wright-Patterson AFB, OH. Written in 2010.

[KoM04] J. Z. Kolter and M. A. Maloof. “Learning to detect malicious executables in the wild,” *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. pp. 479-478, New York: ACM, 2004.

[KoM06] J. Z. Kolter and M. A. Maloof. “Learning to detect and classify malicious executables in the Wild,” *The Journal of Machine Learning Research*. Vol. 6, pp. 2721-2744, 2006.

[LeM06] T. Lee and J. J. Mody. “Behavioral classification,” *2006 EICAR Conference.* 2006.

[LWF09] G. L'Huillier, R. Weber and N. Figueroa. “Online phishing classification using adversarial data mining and signaling games,” *ACM SIGKDD Explorations Newsletter*. Vol. 11, No. 2, pp. 92-99, New York: ACM, 2009.

[Mal10] “MailItem Object.” Retrieved on 25 June 2010 from: http://msdn.microsoft.com/enus/library/aa210946(v=office.11).aspx.

[Mal11] “Malware Domain List: Downloadable Lists,” Retrieved on 10 January 2011 from: http://www.malwaredomainlist.com/forums/index.php?topic=3270.0.

[MeM11] D. Merritt and B. Mullins. “Identifying Cyber Espionage: Towards a Synthesis Approach,” pending publication in *Proceedings of the 6th International Conference on Information Warfare and Security*. 2011.

[Mes08] E. Messmer. “Cyber espionage seen as growing threat to business, government,” *Network World*. 17 January 2008. Retrieved on 10 June 2010 from: http://www.networkworld.com/news/2008/011708-cyberespionage.html.

[MKK07] A. Moser, C. Kruegel and E. Kirda. “Limits of Static Analysis for Malware Detection,” *23rd Annual Computer Security Applications Conference (ACSAC 2007).* pp. 421-430, 2007.

[Off10] “Offensive Computing: Malware Search,” Retrieved on 20 December 2010 from: http://www.offensivecomputing.net.

[RaS02] F. L. Ramsey and D. W. Schafer, “The Statistical Sleuth : A Course in Methods of Data Analysis, 2nd Edition,” Publisher: Brooks/Cole Cengage Learning, ISBN-10:0534386709, pp. 47, 221, 543, and 552-565, 2002.

[Rep09] “Report: Data-stealing Malware Leads to Rise in Cybercrime, Cyberterrorism,” *DarkReading*. 29 June 2009. Retrieved on 10 June 2010 from: http://www.darkreading.com/insiderthreat/security/cybercrime/ showArticle. jhtml?articleID=218101832.

[Rpr11] R Project for Statistical Computing, Retrieved 10 January 2011 from http://www.r-project.org.

[RuS04] M. E. Russinovich and D. A. Solomon. *Microsoft Windows Internals, Fourth Edition: Microsoft Windows Server(TM) 2003, Windows XP, and Windows 2000 (Pro-Developer)*. Microsoft Press, Redmond, WA, USA, 2004.

[SAN08] SANS Institute. "Top Ten Cyber Security Menaces for 2008," Retrieved on 10 June 2010 from: http://www.sans.org/2008menaces/.

[SCY05] D. H. Shih, H. S. Chiang and C. D. Yen. "Classification methods in the detection of new malicious emails," *Information Sciences.* Vol. 171, pp. 241-261, 2005.

[SEZB01] M. G. Schultz, E. Eskin, E. Zadok, M. Bhattacharyya and S. J. Stolfo. "MEF: Malicious Email Filter: A UNIX mail filter that detects malicious windows executables," *Proceedings USENIX Annual Technical Conference: FREENIX Track*. pp. 245-252, Berkely, CA: USENIX Association, 2001.

[SEZS01] M. G. Schultz, E. Eskin, F. Zadok and S. J. Stolfo. "Data mining methods for detection of new malicious executables," *Proceedings of the 2001 IEEE Symposium on Security and Privacy*. pp. 38-49, Washington, D.C.: IEEE Computer Society, 2001.

[SHW+03] S. Stolfo, S. Hershkop, K. Wang, O. Nimeskern and C. W. Hu. "Behavior profiling of email," *Proceedings of the 1st NSF/NIJ Conference on Intelligence and Security Informatics.* pp. 74-90, Berlin: Springer-Verlag, 2003.

[SIK+05] S. Sidiroglou, J. Ioannidis, A. D. Keromytis and S. J. Stolfo. "An email worm vaccine architecture," *Information Security Practice and Experience: Lecture Notes in Computer Science.* Vol. 3439, pp. 97-108, Berlin: Springer, 2005.

[SKF08] M. Shafiq, S. Khayam and M. Farooq. "Embedded malware detection using Markov n-grams," *Proceedings of the 5th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment.* pp. 88-107, Berlin: Springer-Verlag, 2008.

[SWL07] S. J. Stolfo, K. Wang and W. J. Li. "Towards Stealthy Malware Detection," *Malware Detection: Advances in Information Security*. Vol. 27, pp. 231-249, Springer U.S., 2007.

[SWL08] M. Siddiqui, M. C. Wang and J. Lee. "A survey of data mining techniques for malware detection using file features," *Proceedings of the 46th Annual Southeast Regional Conference on XX.* pp. 509-510, New York: ACM, 2008.

[SXC+04] A. Sung, J. Xu, P. Chavez and S. Mukkamala. "Static Analyzer of Vicious Executables (SAVE)," *Proceedings of the 20th Annual Computer Security Applications Conference*. pp. 326-334, Washington, D.C.: IEEE Computer Society, 2004.

[The10] "The InnoDB Storage Engine," Retrieved on 25 September 2010 from: http://dev.mysql.com/doc/refman/5.0/en/innodb-storage-engine.html.

[Top10] "Top Reasons to Use MySQL," Retrieved on 25 September 2010 from: http://www.mysql.com/why-mysql/topreasons.html.

[TSF09] S. M. Tabish, M. Z. Shafiq and M. Farooq. "Malware detection using statistical analysis of byte-level file content," *Proceedings of the ACM SIGKDD Workshop on CyberSecurity and Intelligence Informatics*. pp. 23-31, New York: ACM, 2009.

[USC08] U.S.-China Economic and Security Review Commission, 110th Congress, 2nd Session. *2008 Report to Congress.* Washington, D.C.: GPO, 2008.

[Van08] M. Van Horenbeeck. "Overview of cyber attacks against Tibetan communities," 24 March 2008. Retrieved on 20 December 2009 from: http://isc.sans.edu/diary.html?storyid=4177.

[Vir10] "Virus Bulletin VB100 Archive," Retrieved on 20 May 2010 from: http://www.virusbtn.com/vb100/archive/results?display=platforms.

[VxH10] "VX Heavens," Retrieved on 20 May 2010 from: http://vx.netlux.org/vl.php.

[Wha10] "What is Samba?," Retrieved on 26 September 2010 from: http://www.samba.org/samba/what_is_samba.html.

[WHF07] C. Willems, T. Holz and F. Freiling. "CWSandbox: Towards Automated Dynamic Binary Analysis," *IEEE Security and Privacy*. Vol. 5, No. 2, pp. 32-39, 2007.

[YWL+07] Y. Ye, D. Wang, T. Li and D. Ye. "IMDS: Intelligent malware detection system,: *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. pp. 1043-1047, New York: ACM, 2007.

[ZEC+07] Y. Zhang, S. Egelman, L. Cranor and J. Hong. "Phinding phish: Evaluating anti-phishing tools," *Proceedings of the 14th Annual Network and Distributed System Security Symposium (NDSS 2007).* 2007.

[Zet10] K. Zetter "Google hack attack was ultra sophisticated, new details show," 14

January 2010. Retrieved on 21 November 2010 from: http://www.wired.com/threatlevel/2010/01/operation-aurora/.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED** *(From – To)*
24-03-2011	Master's Thesis	Sep 2009 – Mar 2011

4. TITLE AND SUBTITLE

Spear Phishing Attack Detection

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

David T. Merritt, Capt, USAF

5d. PROJECT NUMBER
ENG 10-391

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT/GCE/ENG/11-05

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Robert Kaufman
318 Information Operations Group/DD
688th Information Operations Wing (AFSPC)
102 Hall Blvd, Suite 311, San Antonio, TX 78243-7078
(210) 925-4425
Robert.Kaufman@us.af.mil

10. SPONSOR/MONITOR'S ACRONYM(S)

318th IOG/DD (688th IOW)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED. This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This thesis addresses the problem of identifying email spear phishing attacks, which are indicative of cyber espionage. Spear phishing consists of targeted emails sent to entice a victim to open a malicious file attachment or click on a malicious link that leads to a compromise of their computer. Current detection methods fail to detect emails of this kind consistently. The SPEar phishing Attack Detection system (SPEAD) is developed to analyze all incoming emails on a network for the presence of spear phishing attacks. SPEAD analyzes the following file types: Windows Portable Executable and Common Object File Format (PE/COFF), Adobe Reader, and Microsoft Excel, Word, and PowerPoint. SPEAD's malware detection accuracy is compared against five commercially-available email anti-virus solutions. Finally, this research quantifies the time required to perform this detection with email traffic loads emulating an Air Force base network. Results show that SPEAD outperforms the anti-virus products in PE/COFF malware detection with an overall accuracy of 99.68% and an accuracy of 98.2% where new malware is involved. Additionally, SPEAD is comparable to the anti-virus products when it comes to the detection of new Adobe Reader malware with a rate of 88.79%. Ultimately, SPEAD demonstrates a strong tendency to focus its detection on new malware, which is a rare and desirable trait. Finally, after less than 4 minutes of sustained maximum email throughput, SPEAD's non-optimized configuration exhibits one-hour delays in processing files and links.

15. SUBJECT TERMS

spear phishing, cyber espionage, malware analysis, malware detection, static analysis, dynamic analysis, email malware

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
REPORT	**ABSTRACT**	**c. THIS PAGE**			Dr. Barry E. Mullins, ENG
U	U	U	UU	128	**19b. TELEPHONE NUMBER** *(Include area code)* (937) 255-3636 x7979; Barry.Mullins@afit.edu

Standard Form 298 (Rev: 8-98)
Prescribed by ANSI Std. Z39-18